CNN For Tomato Leaf Disease Detection

Øystein Haga
Departmet of Computer Science
Oslomet
Oslo, Norway
oyhag8780@oslomet.no

Dung Thuy Vu
Departmet of Computer Science
Oslomet
Oslo, Norway
duvu35218@oslomet.no

Rumman Chowdhury
Department of Computer Science
OsloMet
Oslo, Norway
rucho8446@oslomet.no

Group Number: 5

Abstract

Tomatoes are a widespread, well-known vegetable all over the world and an important crop. Leaf disease of this plant can give us economical loss both to the poor farmers and government. Hence, early detection of tomato plant leaf disease is utmost important. Manual approach to identify disease is error-prone, time-consuming and laborious. Recently, deep learning showed impressive results for image identification and classification. We used 7 layers of Convolutional Neural Network (CNN) with Global Average Pooling and Max Pooling Layer to identify tomato leaf disease. The proposed model gave accuracy of 93.75% with a precision of 90.72% and recall of 87.50%, for the dataset with 10000 images of 9 different leaf diseases and 1 group of healthy leaves.

1. Introduction

Tomatoes, being one of the most widely used vegetables globally, serve as a vital component of healthy diets in numerous forms - consumed raw or prepared in a variety of ways. Recognized for their richness in essential vitamins A and C, tomatoes contribute significantly to maintaining eye health and shielding against sun-induced damage.

On the other hand, the tomato crop is vulnerable to a variety of diseases. There are more than 200 known diseases, and they can have a staggering impact, with up to 70% to 95% loss in yield [9]. The agricultural sector thus faces substantial challenges in preserving crop health and ensuring optimal yield.

Manual detection of plant disease has some limitations: it is time-intensive and susceptible to errors. Carefully inspecting each plant's leaves, identifying diseases, and subsequently administering appropriate treatments to mitigate their spread is a laborious process that consumes valuable time and has potential for misdiagnosis and incomplete treatment.

The primary objective of this project is to use image classification technology for detection of diseases in tomato leaves, by utilizing convolutional neural networks. Disease classification may help global food scarcity, while also saving the income of individual farmers.

1.1 Literature Review

In a review of advancements in ML and DL techniques for plant disease detection between 2015 and 2022, they emphasized that the technology has greatly improved the accuracy and efficiency of plant disease detection, but pointed out that most research has been limited to lab-based studies, and proposed that images from various growth stages, seasons and regions is crucial [1]. Despite acknowledging the significance of diverse image datasets encompassing various growth stages, seasons, and geographical regions in the enhancement of plant disease detection models, our current study focuses on utilizing images captured within an artificial environment. This approach ensures control over environmental variables, thereby facilitating a systematic examination of the model's performance under standardized conditions, fostering a foundational understanding before extrapolating the developed methodologies to real-world scenarios with diverse image datasets.

Another paper addresses CNNs challenge in dealing with small datasets by proposing the method PiTLiD, specifically developed for small sample sizes [2]. It is based on the CNN Inceptionv3 and utilizes transfer learning, meaning knowledge learned from one task is re-used to boost performance in a related task. In our current study, we are taking a simpler approach to deal with limited dataset, by utilizing the data augmentation techniques of flipping, scaling, cropping and rotation.

To address the issue that most datasets consist of images taken in the lab, [3] presents pictures of tomato leaves taken in a real-world environment.

The researchers in [4] wanted to see if they could achieve results similar to state-of-the-art techniques using the relatively simple approach of convolutional neural networks and limited computational power; they reached an average accuracy of 94-95 %. This indicates that high accuracy is achievable even with limited computational power and a relatively simple technique.

Another group of researchers utilized the pre-trained deep learning architectures of AlexNet and VGG16 net [5]. Using 13 262 images, they reached an accuracy of respectively 97.49% and 97.29%, while AlexNet had lower execution time. A deep convolutional neural network was utilized in [6] to perform symptom-wise detection for cucumber disease. The researchers in [7] compared several deep learning architectures for plant disease detection and found that DenseNets reaches an accuracy of 99.75%. Another paper demonstrates the importance of depth in convolutional neural networks by showing that the representation depth is beneficial for the classification accuracy [8].

2. Methods

We are using convolutional neural network (CNN) with Global Average Pooling (GAP) and max pooling layer. CNNs are widely used in computer vision tasks because of their ability to learn and extract hierarchical features from visual data, making them highly effective in tasks related to image analysis.

2.1. Dataset

The dataset consists of pictures of tomato leaves that are either healthy or have one of several diseases. The categories are then:

- Tomato mosaic virus
- Target Spot
- Bacterial Spot
- Tomato Yellow Leaf Curl Virus
- Late Blight
- Leaf Mold
- Early Blight
- Spider Mites – Two spotted Spider Mite
- Septoria Leaf Spot
- Healthy

There are 1000 pictures for each category, so 10 000 pictures in total. The pictures are taken in an artificial environment.

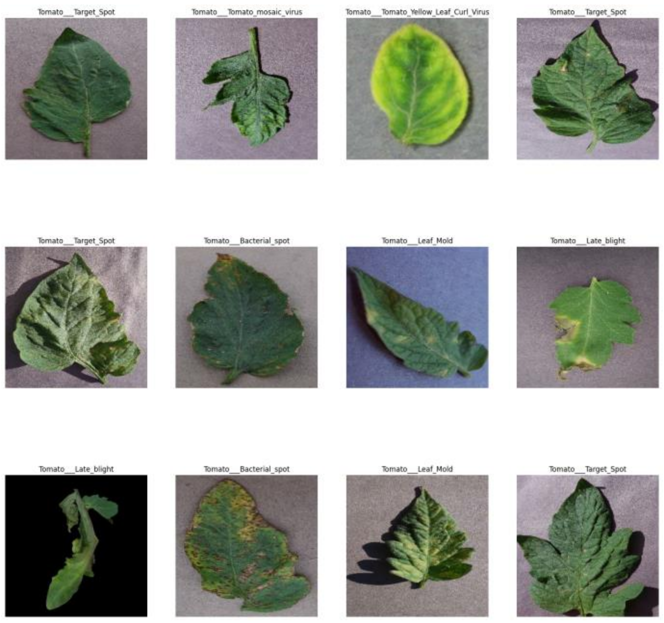


Figure-1: Visualizations of dataset

2.2 Data Pre-Processing

In the realm of tomato disease detection using Convolutional Neural Networks (CNNs), a meticulous and robust data processing framework was carefully devised to optimize the dataset, laying the foundation for subsequent model training and evaluation. The primary objective of this comprehensive approach is to ensure that the CNN model is supplied with high-quality, diverse, and standardized data, thereby enhancing its ability to accurately detect and classify various diseases that affect tomatoes.

The process began by loading the dataset from the 'tomato' directory using TensorFlow's image_dataset_from_directory function. This function enabled the ingestion of batches, each containing 32 images that were uniformly sized at 256x256 pixels. A batch size of 32 indicates that during each training iteration, the neural network updates its weights based on 32 samples simultaneously

The shuffle parameter was activated during loading to ensure that the model was presented with a randomized order of examples during each training epoch. This strategic randomization is essential in preventing the model from learning patterns based on the order of the data.

Moving forward, the dataset underwent partitioning into training, validation, and test sets. This partitioning was facilitated by the get_dataset_partitions_tf function, which ensured a balanced distribution of diverse data across different stages of model development. Such strategic partitioning is essential for training a model that can generalize well to previously unseen data, a critical requirement for CNN's success in real-world applications.

Following the partitioning process, each subset of the dataset underwent a series of optimizations aimed at improving training efficiency. Caching, shuffling, and prefetching were meticulously implemented to expedite data loading, diversify the order of data presentation during training, and parallelize data loading with model execution, respectively. Caching involves storing elements of the dataset in memory after the initial loading. This is particularly advantageous for speeding up data loading, as subsequent iterations can utilize the cached data instead of reloading from the original data source. In our specific context, where a large amount of image data is involved, which is 10000 images, caching becomes essential in reducing the time spent reading data from disk by keeping frequently accessed elements readily available in memory.

In addition to that, shuffling plays a vital role in preventing the model from learning patterns based on the data's order. By randomizing the order of examples within each epoch, the model is exposed to a diverse set of samples, reducing the risk of overfitting to specific patterns associated with the initial presentation order of the data. This is especially important for image classification tasks of tomato disease detection in this analysis, where patterns may emerge based on the sequential arrangement of the data.

Prefetching involves overlapping the data loading and model execution processes. During training, while the model is processing one batch, the input pipeline is simultaneously preparing the next batch in parallel. This parallelization minimizes the idle time of the model, resulting in more efficient training. In the context of image classification, where the dataset can be large, prefetching becomes crucial in optimizing the overall training time.

The dataset underwent further refinement in preparation for ingestion by the neural network through a sequential processing step. In this step, all images were resized to a standardized 256x256 pixel size and the pixel values were normalized to a range of (0, 1). This diligent standardization ensures uniformity in input dimensions and contributes to stabilizing and accelerating the training process, effectively mitigating numerical instability. Equivalently, this method improves the overall effectiveness and dependability of the training procedure, promoting a stronger and more robust approach to the neural network model.

2.3 CNN Architecture

In the model architecture designed for tomato disease detection, the process begins with Layer 0. In this initial step, the input images undergo important preprocessing through the resize_and_rescale function. The purpose of this step is to standardize the input dimensions to a consistent size of 256x256 pixels and rescale the pixel values between 0 and 1 as presented in the data processing part of the analysis. This standardization is crucial as it establishes a uniform scale that is necessary for the subsequent feature extraction process.

Moving on to Layers 1-6, the convolutional layers play a systematic role in analyzing the input images. These layers utilize 3x3 filters(kernels) and the Rectified Linear Unit (ReLU) activation function to convolve across the images. Each layer in this sequence progressively learns hierarchical features that are essential for identifying intricate patterns associated with different tomato diseases. The convolutional layers, from Layers 1 to 6, are fundamental in extracting features by capturing subtle details and textures that are indicative of specific diseases.

Following each convolutional layer, there are max-pooling layers. Mathematically, the process of max-pooling involves dividing the input into non-overlapping rectangular regions and selecting the highest value from each region. The formal expression for max pooling can be defined as follows:

$$\text{MaxPooling}\,(x,\, i,\, j) \;=\; \max_{(\text{a,b})}\; \in \text{Pool Region}\; x_{i+a,\, j+b}$$

The operation extracts the maximum value from the pooling region. The pooling window then moves to the next non-overlapping region, and this process continues until the entire input is covered. These layers strategically reduce the spatial dimensions of the input while retaining important information. By doing so, they focus on the most relevant features and enhance the computational efficiency of the model. The process of max-pooling also contributes to translation invariance, which strengthens the robustness of the learned features and optimizes the overall efficiency of the model.

The purpose of Relu layers is to introduce non-linearity by replacing negative numbers with zeros, enabling it to recognize intricate patterns in the data and facilitating the optimization of the neural network through backpropagation.

Mathematically speaking, the ReLU function can be expressed as f(x) = max(0, x). In simpler terms, this means that for any positive input, the function will return the same value, while for any negative input, it will return zero. Visually, this translates to a piecewise linear function where positive inputs result in a straight line and negative inputs yield zero. This step in our model architecture ensures mathematical stability and enhances the model's ability to recognize complex patterns. The non-linear activation function used in these layers is crucial as it enables the model to uncover intricate relationships within the data.

Layer 7, known as the flatten layer, serves as a transition from the convolutional layers to the dense layers. It converts the learned feature maps into a one-dimensional vector. In the Dense layer, each neuron is connected to all the outputs from the previous layer, allowing the model to evaluate the significance of different learned features when making predictions. In the context of tomato disease detection which implies multi-class classification problems, it is a common practice to use softmax in the final classification layer. Softmax is an activation function that transforms the raw output of the neural network into probability distributions across ten classes of disease. Correspondingly, softmax assigns probabilities to each class, ensuring that the sum of these probabilities is equal to 1. Finally, models will make a definitive decision by selecting the class with the highest probability as the predicted class for tomato diseases.

Following the 65 epochs of learning curve visualization both on loss and accuracy trend, we identified model overfitting. To address this issue, we adopt a proactive approach by making necessary adjustments to our first model. Our aim is to prevent overfitting and improve the overall accuracy of our tests. This iterative process involves fine-tuning hyperparameters, making changes to the architecture, and incorporating regularization techniques. By doing so, we ensure that our model remains robust and effective when dealing with new and unseen data. Our ultimate goal is to develop a well-balanced and highly generalized model that not only performs exceptionally well on the training set but also delivers superior results on previously unseen test data.

2.3.1. The application of fine-tuning parameter

By implementing an Exponential Decay schedule for the learning rate, we can greatly enhance the training process. Initially set at 0.001, the learning rate gradually decreases over time based on the specified schedule. With decay steps of 10,000 and a decay rate of 0.9, the learning rate decreases by 10% at each step, creating a staircase-like pattern. This pattern is achieved by setting the 'staircase' parameter to True. To dynamically adjust the learning rate according to the Exponential Decay schedule, the Adam optimizer is chosen. Known for its adaptive learning rate capabilities, the Adam optimizer ensures that the model's learning rate is optimized throughout the training process. Once the model is compiled, it incorporates the customized optimizer, a sparse categorical cross entropy loss function suitable for multi-class classification, and accuracy as the evaluation metric. The adjustment of the learning rate plays a crucial role in preventing overfitting and enables the model to adapt more effectively to the diverse patterns introduced by the augmented data. As the model encounters various nuances, the decreasing learning rate facilitates finer weight adjustments, leading to smoother convergence and enhanced generalization to unseen data.

2.3.2 The application of Batch Normalization

Batch Normalization is a vital technique in Convolutional Neural Networks which plays a significant role in ensuring model stability, training speed, and overall performance. The second model in this analysis integrates Batch Normalization into the CNN architecture to enhance its capabilities. The main function of Batch Normalization is to normalize the input of each layer, which helps address the internal covariate shift problem during training. As a result, the reduction of internal covariate shifts aids in preventing overfitting by maintaining consistent input distributions.

By standardizing the values within a batch, it reduces the model's sensitivity to initial weights and speeds up convergence. In this architecture, Batch Normalization is strategically placed after each convolutional layer, which includes activation functions like ReLU, and before Max pooling layers. This positioning ensures that the input to subsequent layers remains well-conditioned, making the learning process more effective.

Moreover, Batch Normalization acts as a regularizer, adding to the efficiency of other regularization techniques such as dropout and L2 regularization. It achieves this by reducing the internal covariate shift and introducing some noise during training. By keeping the model from overfitting, the training set, this noise injection helps the model generalize better to new, unobserved data.

2.3.3 The application of Global Average Pooling

The Global Average Pooling layer is employed in place of the fully connected layer in the second model, which is thought to be a revolution of the traditional CNN paradigm and an improvement over the first one, to assess how effective it is in lessening the impact of overfitting. This global average pooling layer computes the average for each feature map, resulting in a singular value per feature map. These averaged values then serve as input for the final classification layer.

The benefits of Global Average Pooling are numerous. One of the key advantages of GAP is its ability to reduce parameters compared to traditional fully connected layers, promoting better generalization to unseen data and mitigating the bias that can arise from overfitting. More precisely, we may observe the transformation that occurs when applying Global Average Pooling (GAP) from an input size of 32x4x4x64 to 32x64. Prior to the application of GAP, the feature maps have dimensions of 4x4 for the spatial dimensions (height and width) and 64 for the depth (number of channels). The initial 32 represents the batch size. Each of the 64 channels in the 4x4 feature maps contains spatial information that has been learned by CNN during the convolutional and pooling operations. GAP involves taking the average value for each channel across the spatial dimensions. In this case, the averaging is performed across the 4x4 spatial grid for each of the 64 channels. Consequently, a single value is obtained for each channel, effectively reducing the spatial dimensions to 1x1. Therefore, after the application of GAP, the dimensions become 1x1x64. Considering the batch size of 32, the final output shape is 32x64. This transformation is crucial to the global pooling process as it condenses the spatial information into a global context. This facilitates the subsequent dense layers in the network to make high-level abstractions and predictions based on the aggregated information from the entire feature maps.

Additionally, GAP provides valuable insights into the contributions of filters within the CNN architecture. By calculating the average values of feature maps, GAP highlights the most important information in each map, offering a clearer understanding of the essential features that influence the classification process. This interpretability is particularly useful in our analysis where model insights are also important as predictive accuracy.

Compared to the traditional fully connected layer, GAP, which maps all features into a single vector without considering the input image size, demonstrates adaptability to varying input image sizes. This adaptability ensures that the model remains flexible in handling diverse datasets with different dimensions.

2.3.4 The application of Random Dropout ratio

The strategic placement of the dropout layer in the model is a crucial aspect of the architecture. In the second model, it follows a dense layer with 64 units and precedes the final dense layer for classification. During the training phase, the dropout technique is employed, which randomly sets a fraction of the input units to zero at each update. This approach prevents co-adaptation of hidden units and promotes better generalization. Moreover, the dropout layer serves as a regularization technique, introducing randomness during training to prevent reliance on specific neurons.

The dropout ratio is a hyperparameter that requires careful tuning to strike the right balance between preventing overfitting and retaining model capacity. If the dropout ratio is too high, the model might struggle to learn useful features, leading to underfitting. On the other hand, if the dropout ratio is too low, the model might not effectively prevent overfitting. The chosen dropout ratio of 0.3 in the model architecture strikes a balance between introducing sufficient randomness for regularization while still retaining an adequate number of units to capture essential patterns. Furthermore, we may undertake hyperparameter tweaking by experimenting with various dropout ratios and evaluating the model's performance on a validation set in order to find the best balance. However, the function for drop out ratio hyperparameter tuning will not be executed due to the computational burden associated with the large data set of 10,000 tomato leaves images.

2.3.5 Enhancing Model Robustness: The Strategic Integration of L2 Regularization

The Convolutional Neural Network for tomato disease detection has been enhanced with regularization techniques, specifically L2 regularization, in the dense layers. In general, regularization is a fundamental concept in deep learning that aims to prevent overfitting. Specifically, L2 regularization, also known as weight decay, is a technique that penalizes large weights in the model by adding a term proportional to the square of the weights to the loss function.

Mathematically, the formula is shown below:

$$L2_{\text{Regularization}} = \lambda \, \Sigma_i \omega_i^2$$

where λ is the regularization coefficient, or strength, that regulates how much the regularization term affects the total loss. The goal of training is to minimize the sum of the standard loss and the L2 regularization term (cross-entropy).

In the case of our analysis about tomato disease classification, the application of L2 regularization to the first dense layer with a strength of 0.01 implies that the optimization process not only minimizes the classification error but also discourages the model from relying heavily on any single feature. This prevents the network from becoming overly complex and overly sensitive to the training data, promoting better generalization to diverse tomato leaf patterns.

By introducing this penalty, the model is encouraged to distribute the importance of features more evenly, thus avoiding over-reliance on specific characteristics that may be prevalent in the training set but less representative of the broader class of tomato diseases. Consequently, it allows contribution to a more robust and generalized tomato disease detection model.

2.3.6 Installation of Early Stopping

Early Stopping is a crucial regularization technique in deep learning, particularly for Convolutional Neural Networks, to prevent overfitting during training. In the given configuration of model 2, the chosen metric for monitoring is 'val_accuracy,' which represents the accuracy on the validation set. The 'max' mode indicates that training will stop if there is no improvement in this accuracy metric. The 'patience' parameter, set to 20, means that training will cease if there is no improvement observed for 20 consecutive epochs. Careful monitoring is essential to avoid overfitting and ensure optimal performance on unseen data represented by the validation set. The 'restore_best_weights' setting is also crucial in maintaining the model's integrity. When set to True, it reverts the weights to the configuration that yielded the best accuracy on the validation set. This step is pivotal in guarding against potential fluctuations in the training process and preserving a model state that generalizes well to new, unseen data. The 'verbose' parameter, set to 1, provides informative messages during training, keeping the user informed about Early Stopping decisions. In essence, Early Stopping acts as a safeguard by dynamically adapting the training duration based on the model's learning progress. By carefully monitoring validation accuracy and intelligently deciding when to stop training, Early Stopping helps build models that excel not only on the training data but also demonstrate robust performance on previously unseen datasets.

2.4 The adjustment of data augmentation

Data augmentation is a fundamental technique used in deep learning, particularly in image classification models, to address the issue of overfitting. Data augmentation artificially expands the training dataset by applying various transformations to the original images. These transformations introduce diversity and variability, preventing the model from memorizing specific patterns and encouraging it to learn more robust features. In the case of the second model for tomato disease detection, common augmentation techniques include random rotations, flips, zooms, shifts, and changes in brightness are used. By incorporating these variations, the model is exposed to a wider range of examples, enhancing its ability to handle different angles, lighting conditions, and perspectives. All of those techniques should be examined in detail respectively. “RandomFlip” randomly flips images both horizontally and vertically. It simulates variations in the orientation of tomato leaves as relocation of mirror images because leaves in real-world scenarios can have different orientations.

“RandomlyRotation” rotates images by a maximum angle of 0.4 radians. It simulates the natural variability in the orientation of tomato leaves, enabling the model to learn features from leaves at different angles. “RandomZoom” with parameter of 0.4 allows random zooming with maximum factor 0.4. It introduces variability in the scale of the leaves, helping the model generalize well to leaves of different sizes. Additionally, “RandomContrast” has been applied for the adjustment of image contrast by the factor of 0.4. It enhances the model's ability to recognize patterns under different lighting conditions. “RandomlyTranslation” translates images both vertically and horizontally by value of 0.1. It simulates slight variations in the positioning of leaves within the images, contributing to the model's ability to handle different spatial arrangements. The combination of those methods allows a better model with a more comprehensive and diverse set of training examples, reducing the risk of overfitting. Moreover, the increased variability in the augmented dataset helps the model generalize well to unseen data, ultimately improving its performance in detecting tomato diseases under various conditions.

3. Experiments and results

3.1 Experimental Setup

3.1.1 Hardware and software configurations

This project focuses on utilizing the TensorFlow framework and Google Colab to detect diseases in tomato plants through deep learning techniques. By harnessing the capabilities of convolutional neural networks (CNNs), our objective is to create a robust model that can effectively identify and classify various diseases that impact tomato crops. TensorFlow offers a flexible and user-friendly environment for implementing intricate neural network architectures, while Google Colab provides the added benefit of free access to GPU acceleration via Google's cloud infrastructure, facilitated by the GPY library. The utilization of GPUs greatly enhances the training process, enabling quicker experimentation and refinement of the model.

The nvidia-smi command is utilized in this project to obtain GPU information and verify the connection to a GPU. Following this, the script attached for this project configures the virtual device through TensorFlow to check and limit GPU memory usage if necessary. The code employs mixed precision by utilizing the TensorFlow Mixed Precision API to enhance training efficiency and reduce memory consumption by using lower precision (float16) for specific computations. These configurations aim to optimize computational resources, ensuring efficient GPU utilization and improved performance for the tomato disease detection models.

3.1.2 Evaluation Metrics

The first evaluation metrics that have been used for this tomato disease detection project is learning curves. Learning curves provide nuanced insights into the model's training dynamics. The accuracy learning curve indicates the model's ability to learn from the training data over a variety of epochs, with an upward trend indicating successful learning. A steep increase signals rapid convergence, while a plateau may suggest that the model has reached its learning capacity. In contrast, the loss curve illustrates the diminishing prediction errors during training, with a rapid decline signifying efficient convergence. However, an erratic or fluctuating trend could indicate issues such as learning rate instability. Discrepancies between the training and validation accuracy curves require careful consideration. If the training accuracy surges while the validation accuracy lags, the model may be overfitting the training data, capturing noise rather than true patterns. Conversely, if both curves converge but exhibit suboptimal accuracy, the model might be underfitting, necessitating a more complex architecture or additional training epochs. The loss curves on both datasets provide similar insights into the model's convergence.

Additionally, observing the learning curves on the testing dataset is crucial for assessing the model's generalization to novel data. A consistent trend across all datasets indicates robust performance, while erratic behavior on the testing dataset may signal issues with generalization. As a component of this analysis, the performance trajectories of both models will be visually represented by a graph that illustrates the trends in test accuracy across 65 epochs. The purpose of this graphical representation is to facilitate a direct comparison between the two models in terms of testing accuracy, highlighting their respective strengths and areas for improvement.

The second metric used in this analysis is confusion matrix. This confusion matrix offers a comprehensive overview of our model's classification efficacy by encapsulating the interplay between predicted and true classes. Each cell in the matrix corresponds to a specific class, delineating instances of true positives, true negatives, false positives, and false negatives. The diagonal elements signify accurate predictions, while off-diagonal elements pinpoint misclassifications. This metric is crucial in evaluating the model's ability to distinguish between different disease categories in tomato plants. High values along the diagonal represent successful disease identifications, while deviations suggest areas for improvement.

Examining and interpreting false positives (FP) and false negatives (FN) helps us understand specific misclassifications, providing actionable insights for model refinement. False positives (FP), indicating instances where the model predicted a disease incorrectly, may lead to unnecessary interventions for healthy plants. In contrast, false negatives (FN), representing cases where the model failed to detect a disease, can be critical, potentially leading to untreated diseased plants. Overall, the confusion matrix used in the analysis provides a nuanced perspective on the model's performance comparison, facilitating informed decisions for optimization and ensuring the robustness of the classification outcomes of 10 classes of tomato disease.

From the confusion matrix we can derive several performance metrics:

$$\text{recall} = \frac{TP}{TP + FN}$$

Recall, also referred to as sensitivity or true positive rate, measures the model's capacity to accurately identify all instances of a specific class, particularly those affected by diseases within our given context. A high recall value indicates that the model effectively minimizes the chances of missing diseased plants, which is of utmost importance in the field of agriculture where timely detection plays a vital role in disease management. Conversely, a low recall value suggests an increased risk of overlooking diseased plants, potentially leading to untreated crops. On the other hand, precision evaluates the accuracy of positive predictions made by our model, indicating the proportion of correctly identified positive instances out of all instances predicted as positive. Its formula is

$$\text{precision} = \frac{TP}{TP + FP}$$

with *FP* referring to false positives.
In the context of detecting tomato diseases, precision holds significant importance in minimizing false positives, as misclassifying a healthy plant as diseased may result in unnecessary interventions. A high precision value indicates a reduced likelihood of making false-positive predictions, thereby contributing to the precision of agricultural practices. It not only enhances cost-effectiveness but also encourages eco-friendly and mindful approaches to controlling diseases in tomato farming.

In our analysis, finding the right balance between recall and precision is crucial. Achieving a high recall ensures the effective identification of diseased plants, while maintaining a high precision guarantees the reliability of the model's positive predictions. This equilibrium, evaluated through metrics like the ROC curve and the elbow point, will be a key criterion for assessing the performance of the two models developed in this analysis. Striking this balance is essential for optimizing the reliability, accuracy, and economic viability of our tomato disease detection system.
The **F1 score** offers a balanced assessment of the model’s performance, by combining precision and recall in the following formula:

$$\text{F1-score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

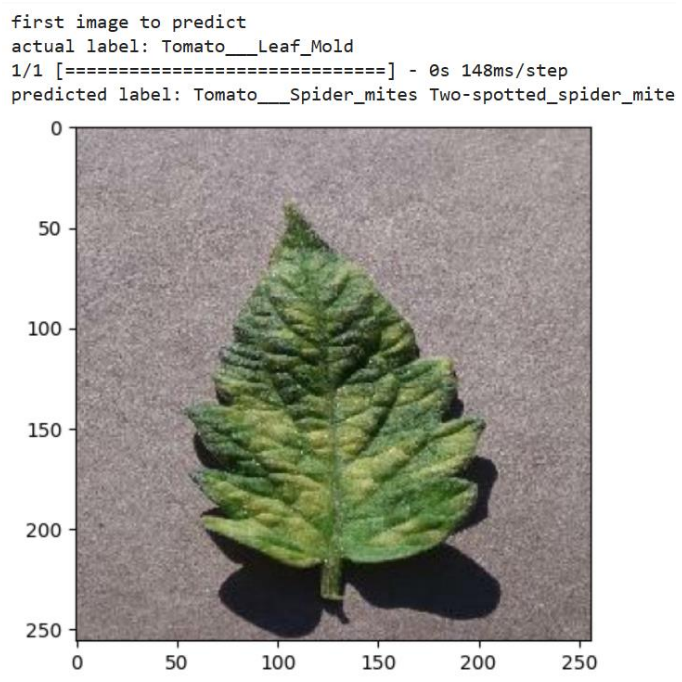
4. Results and Discussion

4.1 Test Image Evaluation

In this part, some random test images have been evaluated. The function for prediction is used to obtain the prediction probabilities for each class. The class with the highest probability is considered as the predicted class, and the confidence level is calculated as a percentage based on this maximum probability. The resulting visualization shown below includes the actual class, predicted class, and confidence level for each image, providing a comprehensive view of the first model performance on the test dataset.

The proficiency of the first model was evident in the examination of 9 test images, as all of them displayed correct predictions for their respective true diseases. It's pertinent to observe that the confidence intervals associated with these predictions were notably high, with 8 out of the 9 images hovering around 99%. This high confidence level indicates the robustness and confidence ability to accurately identify specific diseases in tomato plants. However, one test image, which corresponded to tomato mosaic viruses, achieved a slightly lower confidence interval of 86.08%. While still substantial, this lower confidence level suggests a comparatively higher degree of uncertainty in the prediction. Because the confidence interval serves as a measure of the certainty for that prediction. A higher confidence interval among the other 8 images indicates a stronger belief in the correctness of the assigned class, while a lower confidence interval in that tomato mosaic virus suggests a degree of uncertainty.

The image below shown the incorrect prediction when the predicted disease is Two-spotted_spider_mite while the actual label is Tomato_leaf_Mold



```
first image to predict
actual label: Tomato___Leaf_Mold
1/1 [==============================] - 0s 148ms/step
predicted label: Tomato___Spider_mites Two-spotted_spider_mite
```

Figure-2: Incorrect prediction of disease

Based on this initial examination, the model demonstrates promising performance by providing accurate predictions with high confidence for the majority of the test images. However, it is important to acknowledge that a more extensive test set or test accuracy learning curve is necessary for a comprehensive evaluation.

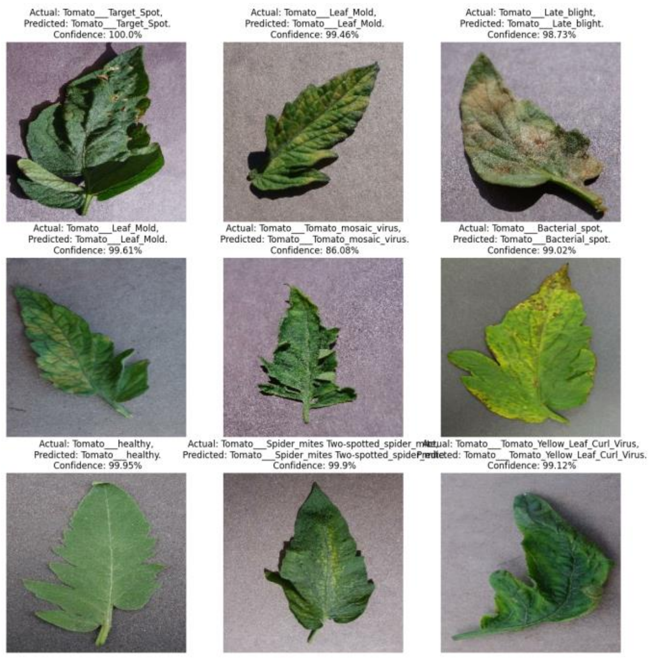

Actual: Tomato__Target_Spot,
Predicted: Tomato__Target_Spot.
Confidence: 100.0%
Actual: Tomato__Leaf_Mold,
Predicted: Tomato__Leaf_Mold.
Confidence: 99.46%
Actual: Tomato__Late_blight,
Predicted: Tomato__Late_blight.
Confidence: 98.73%
Actual: Tomato__Leaf_Mold,
Predicted: Tomato__Leaf_Mold.
Confidence: 99.61%
Actual: Tomato__Tomato_mosaic_virus,
Predicted: Tomato__Tomato_mosaic_virus.
Confidence: 86.08%
Actual: Tomato__Bacterial_spot,
Predicted: Tomato__Bacterial_spot.
Confidence: 99.02%
Actual: Tomato__healthy,
Predicted: Tomato__healthy.
Confidence: 99.95%
Actual: Tomato__Spider_mites Two-spotted_spider_mite,
Predicted: Tomato__Spider_mites Two-spotted_spider_mite.
Confidence: 99.9%
Actual: Tomato__Tomato_Yellow_Leaf_Curl_Virus,
Predicted: Tomato__Tomato_Yellow_Leaf_Curl_Virus.
Confidence: 99.12%

Figure 3: Model Performance with confidence level

4.2 Quantitative Results

Out of 1000 test samples, the model successfully made 896 correct predictions, highlighting its ability to accurately identify various tomato diseases. However, it is important to note that there were some misclassifications represented by the off-diagonal elements in the confusion matrix. For instance, there are 42 misclassifications between Tomato_Target Spot and Tomato_Spider Mites. It implies that the model incorrectly identified some instances of Tomato_Spider_Mites as Tomato_Target_Spot (False Positives), and it failed to identify some instances of Tomato_Target_Spot as Tomato_Target_Spot (False Negatives). The cumulative count of these misclassifications amounts to 42. Other misclassification occurs between Tomato_ Late_ Blight and Tomato_Early_Blight with value of 19, etc.

Model 2 struggled to differentiate between Tomato Target Spot and Tomato Spider Mites. This difficulty may arise due to visual similarities or overlapping features between these two disease classes. This topic will be elaborated on in the subsequent section.

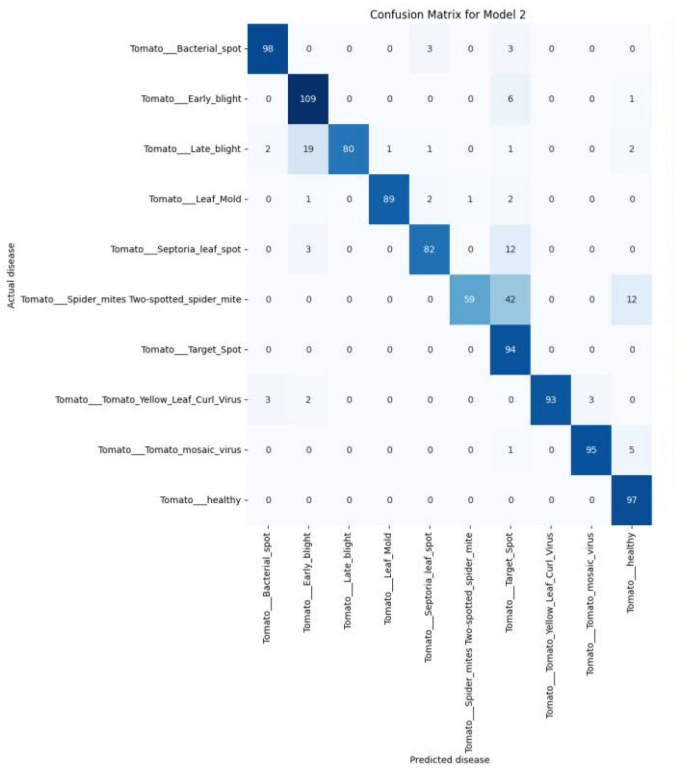


Figure 4: Confusion matrix for Model 2

Precision and Recall

In this analysis, a precision value of 0.9072 indicates that approximately 90.72% of the instances predicted as positive by Model 2 are indeed positive. On the other hand, Recall, also referred to as Sensitivity, measures the ability to identify all positive instances. A recall value of 0.8750 in this analysis indicates that Model 2 successfully captures around 87.50% of all actual positive instances. Regarding the prediction of tomato illnesses, Model 2's greater accuracy score of 0.9072 indicates that, in the vast majority of situations, the model accurately classifies a tomato as diseased, hence boosting confidence in the positive predictions. However, with a recall of 0.8750, the model may overlook a portion of actual diseased tomatoes, indicating a cautious approach in capturing the entirety of diseased instances, possibly to minimize false positives.

We can obviously see that the precision is higher than recall. It implies that the model is cautious in labeling instances as positive. It prioritizes making accurate positive predictions but may be more conservative in capturing all positive instances, resulting in a lower recall. In practical terms, a higher precision and lower recall mean that when the model predicts a positive instance, it is likely to be correct. However, there is a risk of missing some actual positive instances.

Generally speaking, these values indicate a commendable balance between precision and recall, indicating that the model achieves accurate positive predictions while effectively capturing the majority of positive instances.

```
Precision for model 2: 0.9072
Recall for model 2: 0.8750
```

4.3 Comparing the performance of models

Throughout the training process, learning curves for Model 1 display complex dynamics. The accuracy curve exhibits a rapid and smooth rise in training accuracy from epochs 1 to 8 or 9, achieving an approximate accuracy of 83%. Subsequently, there is a gradual increase to 93% at epoch 25. This is followed by a slower ascent, culminating in a peak accuracy of 96% at epoch 50, characterized by minimal fluctuations. On the other hand, the validation accuracy initially matches the training accuracy, reaching up to 70%. However, it diverges thereafter and consistently lags behind, displaying more noticeable fluctuations.

A similar pattern is observed in the loss curve. The training loss demonstrates a significant and smooth decrease, reaching approximately 0.4 by epoch 10. The rate of decrease then slows down, indicating a gradual refinement process. In contrast, the validation loss follows a similar initial trend, starting at 0.8, but introduces fluctuations, suggesting potential challenges in generalization. The mirroring effect between the accuracy and loss curves highlights the model's adaptability to the training data. However, the fluctuations in the validation metrics indicate areas that require improvement.

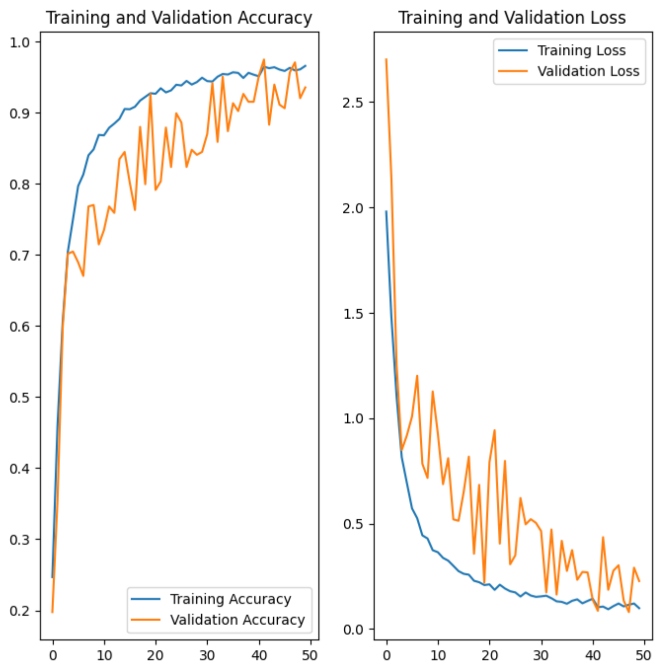


Figure-5: Training accuracy and validation accuracy for model 1

The fact that the training accuracy and validation accuracy for model 1 are closely aligned, with the latter consistently falling behind and displaying fluctuations, suggests that there may be a risk of overfitting. It is often beneficial to strive for a slightly lower validation accuracy, as this indicates that the model is not over-optimizing for the training data and has a better chance of achieving higher accuracy on unseen test data. The methodical adjustments applied to Model 2 have efficaciously alleviated concerns of overfitting, fostering a harmonized performance marked by a convergence between training and validation accuracy as shown below:

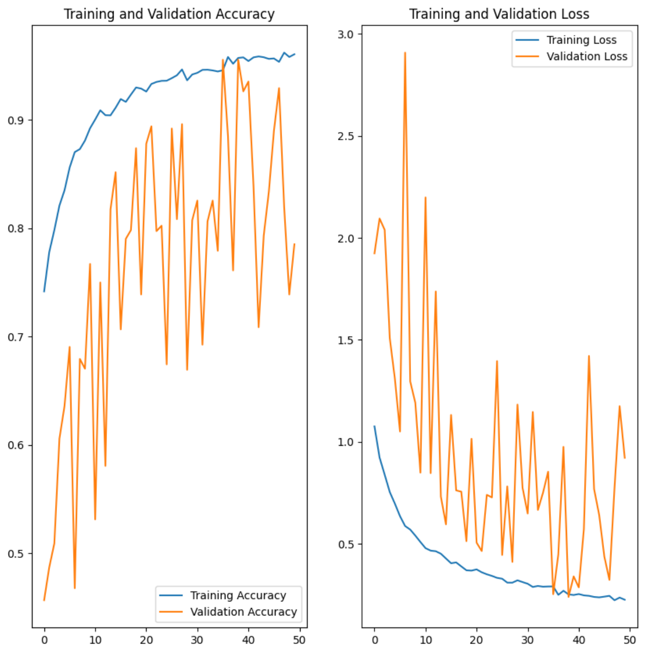


Figure-6: Training accuracy and validation accuracy for model 2

Model 2 displays unique characteristics in its learning curves, providing insight into its training dynamics. The training accuracy starts at 75% in epoch 1, showing a relatively smoother trajectory compared to Model 1 in the initial epochs. As training progresses, the accuracy gradually improves with subtle fluctuations, indicating a steady learning process without any evident obstacles like vanishing gradient descent. Conspicuously, the training accuracy consistently rises, demonstrating a resilient learning trend. In contrast to Model 1, the validation accuracy for Model 2 follows a generally lower path. This difference is noticeable throughout the training process, suggesting that the model faces more challenges in generalizing to unseen data compared to its predecessor due to the adjustment for overfitting tackling. The validation accuracy exhibits more significant fluctuations, peaking at 95% around epoch 35 and then dropping significantly to 75% by epoch 40. These extreme fluctuations highlight the model's sensitivity to variations in the validation dataset.

The loss curve for Model 2 mirrors its accuracy counterpart, revealing an intriguing pattern. The training loss starts at approximately 1.1 at the first epoch and gradually decreases to 0.2 at the last epoch, indicating effective learning and refinement of the model two representations. However, the validation loss diverges, starting at around 2 and showing extreme fluctuations. This substantial variability suggests difficulties in maintaining consistency and generalization on the validation set.

Based on this graph, it is recommended that Model 2 undergo further refinement to improve its generalization capabilities. Strategies such as implementing regularization techniques, adjusting the model architecture complexity which is used successfully address the observed difficulties in overfitting problems.

4.4 Mean Test Accuracy

The increase in mean test accuracy from Model 1 to Model 2 is quite significant. Model 1 performed well with an accuracy of 89.85% over 65 epochs, indicating a strong baseline performance. However, the implementation of techniques to address overfitting in Model 2 resulted in a remarkable improvement, achieving a mean test accuracy of 93.75%. This increase of approximately 3.9 percentage points highlights the positive impact of the adjustments made to mitigate overfitting in Model 2, allowing it to generalize better to unseen data. The fact that Model 2 outperforms Model 1 in mean test accuracy demonstrates the effectiveness of the applied strategies, resulting in a more robust and reliable model. This improvement is particularly noteworthy when considering a dataset of 10,000 images, as it suggests that Model 2 is better equipped to make accurate predictions on a diverse range of instances. It underscores the importance of implementing techniques to combat overfitting, emphasizing their role in enhancing the overall performance and reliability of deep learning models.

```
Mean Test Accuracy for model 1 over 65 epochs: 89.85%

Mean Test Accuracy for model 2 over 65 epochs: 93.75%
```

4.5 Test Accuracy Learning curve

The comparison of test accuracy between Model 1 (blue line) and Model 2 (green line) reveals distinct patterns and considerations. Model 1 consistently demonstrates lower test accuracy throughout the epochs when compared to Model 2. The majority of data points on the blue line exhibit inferior accuracy compared to their corresponding points on the green line. Furthermore, Model 1 displays more prominent fluctuations in test accuracy, indicating a higher sensitivity to variations in the test dataset. Both lines exhibit a similar trend of gradual increase from epoch 1 to approximately epoch 28, followed by a subsequent dip and another rise towards epoch 65. This nonlinear pattern suggests that test accuracy does not consistently improve over epochs, potentially reflecting the models' response to specific characteristics or challenges present in the dataset. The source of this difficulty may stem from the existing obstacles in the field of medical imaging analysis. Those can hinder the accuracy of predictions in this analysis. These obstacles include the presence of various disease types, fluctuations in image quality and conditions, imbalanced data, seasonal effects, and potential inconsistencies in labeling. However, by focusing on improved data curation and making necessary adjustments to the models, it is possible to overcome these challenges and enhance the predictive capabilities across a broad spectrum of conditions and diseases.

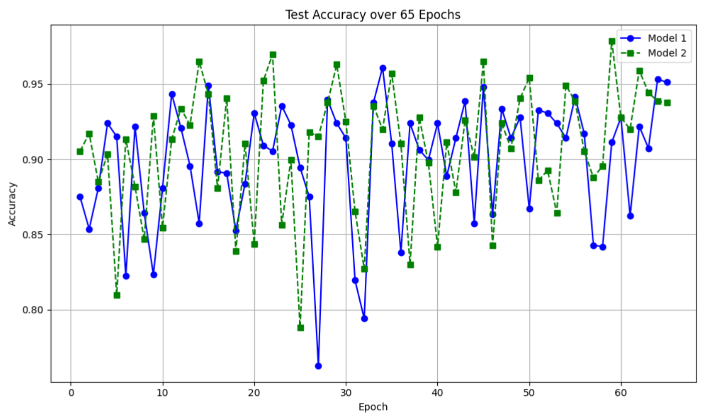


Figure-7: Test accuracy for both model 1 and model 2 over 65 epochs

4.6 Strengths of the Proposed Approach:

Our proposed approach exhibits notable strengths, particularly in its ability to address concerns related to overfitting during the transition from Model 1 to Model 2. By effectively identifying overfitting in Model 1, the approach strategically constructs Model 2, presenting a dynamic and responsive methodology in the face of the challenges posed by deep learning. This iterative refinement of the sequential model leads to a significant enhancement in test accuracy, thereby highlighting the efficacy of the development process. Model 2 in this analysis expertise in diagnosing and mitigating overfitting emphasizes the importance of continuous monitoring and modification in the search of robust deep learning models, both in general and particularly for image classification.

4.7 *Weaknesses and Challenges in the Approach*

Despite these strengths, the proposed approach encounters distinct weaknesses and challenges that necessitate careful consideration. Model 2 faces a formidable obstacle in the form of substantial fluctuations in validation accuracy, resulting in a non-linear trend in test accuracy characterized by oscillations rather than a consistent upward trajectory. The challenge is further compounded when attempting to implement early stopping, as the model consistently runs for 65 epochs regardless of varying patience levels ranging from 20 to 5, and even 4. Even though the initial purpose of installing Early stopping is for promoting better generalization and efficient resource usage, it failed to be applicable in this analysis.

4.8 *Comparison with Previous Studies*

The mean test accuracy for Model 1 and Model 2 were 89.85% and 93.75% respectively. This is comparable to LeNet, which reached an accuracy on tomato leave disease detection of 94-95% [4]. Meanwhile DenseNets reaches an accuracy of 99.75 in one comparative study [7]. This indicates that there is room for improvement. We must be careful in drawing conclusions from this, as they did not use the same dataset.

4.9 Limitations and Future Work

The limitation of this analysis has parallels with challenges encountered in medical data processing which leads to misclassification in confusion matrix.

Just as medical diagnoses require precision, accurately identifying tomato diseases necessitates careful examination. To tackle these similar intricacies, we propose a systematic approach to gain insights into misclassifications as future work should be considered. To improve the accuracy of the tomato disease detection model, it is crucial to conduct a comprehensive analysis of diseases that have high values of misclassifications such as Tomato Target Spot and Tomato Spider Mites or Tomato Late Blight or Tomato Early Blight. The first step involves a meticulous review of misclassified images, specifically focusing on instances where confusion arises between these two diseases. By visualizing these images, we can identify commonalities and differences, which will provide valuable insights into the challenges faced by the model. Attention should be given to specific visual patterns, colors, shapes, or textures that pose difficulties for accurate classification.

Following the image review, a detailed analysis of the features used by the model for classification becomes essential. By extracting and scrutinizing these image features, we can pinpoint the elements that cause confusion between them. This feature analysis is a critical step in understanding the nuances of disease characteristics that contribute to misclassifications. Additionally, it is advisable to consult domain experts in the medical fields or agricultural fields to identify and validate relevant features that effectively distinguish between these diseases.

Overall, this comprehensive methodology, which includes reviewing images and analyzing features, establishes a solid basis for acquiring deep understandings of misclassifications. By utilizing these insights, subsequent modifications to the model can be made, leading to improved accuracy in distinguishing between tomato diseases. Ultimately, this enhances the overall precision and dependability of the tomato disease detection system.

The second limitation in this analysis is the uselessness of Early stopping installation through 65 epochs. This raises concerns regarding the underlying causes of the fluctuating validation accuracy. Several potential factors may contribute to this issue. Firstly, the complexity of the dataset may require a more nuanced approach to data preprocessing in order to stabilize the training signals. Secondly, adjustments to the model architecture might be necessary to enhance its adaptability to the dynamic nature of the dataset we used. Additionally, augmenting the dataset with a more diverse set of training examples could potentially mitigate the fluctuations. It is crucial to carefully examine whether the model is reaching a learning plateau, indicating a need for more epochs, or if the fluctuation is indicative of vanishing gradient descent, which would necessitate a reassessment of the neural network's configuration.

This analysis provides a foundation for suggested improvements in the form of future work, addressing two constraints that have been identified. It also serves as a basis for recommended changes in the form of future work.

5. Conclusions

We have used convolutional neural networks for disease classification in tomato leaves. The dataset consisted of 10 000 pictures divided equally into 10 categories, one of them being healthy plants and the others being various diseases.

In pre-processing the data, the dataset was shuffled to prevent the model from learning patterns based on the order of the data. The dataset was then partitioned into training, validation and testing sets while ensuring balanced distribution. Each subset was optimized to improve the training efficiency. We resized every image to a standardized 256x256 pixel size, with pixel values normalized to a range of (0, 1), which stabilizes and accelerates the training process.

The CNN architecture is built up of several layers. The input is preprocessed in layer 0. In Layers 1-6, the features are learned by the model. In layer 7, the flatten layer, the learned feature maps are converted into a one-dimensional vector, while in the Dense layer, each neuron is connected to all the outputs from the previous layer, allowing the model to evaluate the significance of different learned features when making predictions. In the final layer, we use Softmax to transform the raw output of the neural network into probability distributions across the ten classes of diseases.

Due to signs of overfitting, we adjusted our first model. Exponential Decay schedule for the learning rate was implemented to enhance the training process. We also implemented Batch Normalization, which aids in preventing overfitting by maintaining consistent input distributions. We replaced the fully connected layer with a Global Average Pooling layer, which computes the average for each feature map, resulting in a singular value per feature map that serves as input for the final classification layer. Random Dropout Ratio is employed, which randomly sets a fraction of the input units to zero at each update, preventing co-adaptation of hidden units and promoting better generalization. L2-regularization was implemented, contributing to a more robust and generalized tomato disease detection model. We also implemented Early Stopping, which dynamically adapts the training duration based on the model's learning progress. We also performed data augmentation, which introduces artificial diversity in the dataset, helping to prevent overfitting.

Over the 65 epochs, the mean test accuracy of Model 1 was 89.85%, while for Model 2 it was 93.75%. The Precision and recall for Model 2 was 90.72% and 87.50% respectively, giving an f1-score of 89.08%, indicating good performance.

References

[1] Muhammad Shoaib, Babar Shah, Shaker EI-Sappagh, Akhtar Ali, Asad Ullah, Fayadh Alenezi, Tsanko Gechev, Tariq Hussain Farman Ali, "An advanced deep learning models-based plant disease detection: A review of recent research" Frontiers in Plant Science, vol 14, 2023, https://www.frontiersin.org/articles/10.3389/fpls.2023.1158933

[2] K. Liu and X. Zhang, "PiTLiD: Identification of Plant Disease From Leaf Images Based on Convolutional Neural Network," in *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 20, no. 2, pp. 1278-1288, 1 March-April 2023, doi: 10.1109/TCBB.2022.3195291. https://ieeexplore.ieee.org/document/9847052

[3] Gehlot, M., Saxena, R.K. & Gandhi, G.C. "Tomato-Village": a dataset for end-to-end tomato disease detection in a real-world environment. *Multimedia Systems* **29**, 3305–3328 (2023). https://doi.org/10.1007/s00530-023-01158-y

[4] P. Tm, A. Pranathi, K. SaiAshritha, N. B. Chittaragi and S. G. Koolagudi, "Tomato Leaf Disease Detection Using Convolutional Neural Networks," *2018 Eleventh International Conference on Contemporary Computing (IC3)*, Noida, India, 2018, pp. 1-5, doi: 10.1109/IC3.2018.8530532. https://ieeexplore.ieee.org/document/8530532

[5] Aravind Krishnaswamy Rangarajan, Raja Purushothaman, Aniirudh Ramesh, "Tomato crop disease classification using pre-trained deep learning algorithm," Procedia Computer Science, Volume 133, 2018, https://www.sciencedirect.com/science/article/pii/S1877050918310159

[6] Juncheng Ma, Keming Du, Feixiang Zheng, Lingxian Zhang, Zhihong Gong, Zhongfu Sun, "A recognition method for cucumber diseases using leaf symptom images based on deep convolutional neural network," Computers and Electronics in Agriculture, Volume 154, https://www.sciencedirect.com/science/article/abs/pii/S0168169918309360

[7] Edna Chebet Too, Li Yujian, Sam Njuki, Liu Yingchun, "A comparative study of fine-tuning deep learning models for plant disease identification," Computers and Electronics in Agriculture, Volume

161, 2019, https://www.sciencedirect.com/science/article/abs/pii/S0168169917313303

[8] Karen Simonyan and Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2015, https://arxiv.org/abs/1409.1556

[9] A. N. Lukyanenko, "Disease Resistance in Tomato," https://link.springer.com/chapter/10.1007/978-3-642-84275-7_9

[10] Tomato leaf disease detection (dataset), https://www.kaggle.com/datasets/kaustubhb999/tomatoleaf

Tomatoes Disease Classification

```
# Attack link to Kaggle dataset used
from IPython.display import HTML
HTML('<a
href="https://www.kaggle.com/datasets/kaustubhb999/tomatoleaf"
target="_blank">Click here to access the Kaggle dataset</a>')

# GPU check
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
Wed Nov 29 13:30:53 2023
+-----------------------------------------------------------------------
--------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version:
12.0     |
|-------------------------------+----------------------
+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util
Compute M. |
|                               |                      |
MIG M. |
|
===============================+======================+===============
=======|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |
0 |
| N/A   66C    P8    11W /  70W |      0MiB / 15360MiB |      0%
Default |
|                               |                      |
N/A |
+-------------------------------+----------------------
+----------------------+



+-----------------------------------------------------------------------
--------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name                  GPU
Memory |
|        ID   ID
Usage      |
|
======================================================================
=======|
|  No running processes found
|
+---------------------------------------------------------------------
--------+
```

```
# Code for saving notebook
from IPython.display import display, Javascript

# Function to save the notebook
def save_notebook():
    display(Javascript("google.colab.saveNotebook();"))

# Set up a timer to save the notebook every 15 minutes
save_interval_minutes = 15
save_interval_milliseconds = save_interval_minutes * 60 * 1000

# Save the notebook initially
save_notebook()

# Set up an interval timer to save the notebook automatically
interval_code = f"setInterval(() => {save_notebook()},
{save_interval_milliseconds});"
display(Javascript(interval_code))
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

```
#Installation
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in
/usr/local/lib/python3.10/dist-packages (2.14.0)
Requirement already satisfied: absl-py>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (23.5.26)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1
in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in
```

```
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (16.0.6)
Requirement already satisfied: ml-dtypes==0.2.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy>=1.23.5 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.23.5)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (23.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!
=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.20.3)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.3.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (4.5.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.34.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.59.2)
Requirement already satisfied: tensorboard<2.15,>=2.14 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.1)
Requirement already satisfied: tensorflow-estimator<2.15,>=2.14.0
in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.0)
Requirement already satisfied: keras<2.15,>=2.14.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0-
>tensorflow) (0.41.3)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14-
>tensorflow) (2.17.3)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14-
>tensorflow) (1.0.0)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14-
>tensorflow) (3.5.1)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14-
>tensorflow) (2.31.0)
```

```
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0
in /usr/local/lib/python3.10/dist-packages (from
tensorboard<2.15,>=2.14->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14-
>tensorflow) (3.0.1)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.15,>=2.14->tensorflow) (5.3.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.15,>=2.14->tensorflow) (0.3.0)
Requirement already satisfied: rsa<5,>=3.1.4 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.15,>=2.14->tensorflow) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth-
oauthlib<1.1,>=0.5->tensorboard<2.15,>=2.14->tensorflow) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.15,>=2.14->tensorflow) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.15,>=2.14->tensorflow) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.15,>=2.14->tensorflow) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.15,>=2.14->tensorflow) (2023.7.22)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1-
>tensorboard<2.15,>=2.14->tensorflow) (2.1.3)
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in
/usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1-
>google-auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow) (0.5.0)
Requirement already satisfied: oauthlib>=3.0.0 in
/usr/local/lib/python3.10/dist-packages (from requests-
oauthlib>=0.7.0->google-auth-oauthlib<1.1,>=0.5-
>tensorboard<2.15,>=2.14->tensorflow) (3.2.2)
```

```python
import tensorflow as tf
print("GPU Available: ", tf.config.list_physical_devices('GPU'))
```

```
GPU Available:  [PhysicalDevice(name='/physical_device:GPU:0',
device_type='GPU')]
```

Import all the Dependencies

```python
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from IPython.display import HTML
```

Set all the Constants

```python
BATCH_SIZE = 32
IMAGE_SIZE = 256
CHANNELS=3
EPOCHS=50
```

Import data

```python
# Upload dataset to googlecolab
!pip install opendatasets
!pip install pandas
```

```
Collecting opendatasets
  Downloading opendatasets-0.1.22-py3-none-any.whl (15 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from opendatasets) (4.66.1)
Requirement already satisfied: kaggle in
/usr/local/lib/python3.10/dist-packages (from opendatasets) (1.5.16)
Requirement already satisfied: click in
/usr/local/lib/python3.10/dist-packages (from opendatasets) (8.1.7)
Requirement already satisfied: six>=1.10 in
/usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets)
(1.16.0)
Requirement already satisfied: certifi in
/usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets)
(2023.7.22)
Requirement already satisfied: python-dateutil in
/usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets)
(2.8.2)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets)
(2.31.0)
Requirement already satisfied: python-slugify in
/usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets)
(8.0.1)
Requirement already satisfied: urllib3 in
/usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets)
(2.0.7)
Requirement already satisfied: bleach in
/usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets)
(6.1.0)
Requirement already satisfied: webencodings in
```

```
/usr/local/lib/python3.10/dist-packages (from bleach->kaggle-
>opendatasets) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in
/usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle-
>opendatasets) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->kaggle-
>opendatasets) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests->kaggle-
>opendatasets) (3.4)
Installing collected packages: opendatasets
Successfully installed opendatasets-0.1.22
Requirement already satisfied: pandas in
/usr/local/lib/python3.10/dist-packages (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2023.3.post1)
Requirement already satisfied: numpy>=1.21.0 in
/usr/local/lib/python3.10/dist-packages (from pandas) (1.23.5)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1-
>pandas) (1.16.0)
```

```python
import opendatasets as od

# Provide the Kaggle dataset URL
dataset_url =
'https://www.kaggle.com/datasets/kaustubhb999/tomatoleaf'

# Download the dataset
od.download(dataset_url)
```

```
Please provide your Kaggle credentials to download this dataset. Learn
more: http://bit.ly/kaggle-creds
Your Kaggle username: dungthuyvu
Your Kaggle Key: ··········
Downloading tomatoleaf.zip to ./tomatoleaf

100%|██████████| 179M/179M [00:05<00:00, 35.2MB/s]
```

```python
# Some steps to locate data
import os

# List files in the '/content' directory
files_in_directory = os.listdir('/content')

# Print the list of files
```

```
print("Files in /content:", files_in_directory)

# ok.so it has been succesfully unzip
```

```
Files in /content: ['.config', 'tomatoleaf', 'sample_data']
```

```
import os

# Specify the path to the dataset directory
dataset_path = '/content/tomatoleaf/'  # Adjust based on your dataset

# List files in the dataset directory
files = os.listdir(dataset_path)

# Print the list of files
print("Files in the dataset directory:", files)
```

```
Files in the dataset directory: ['tomato']
```

```
import os

# Specify the path to the dataset directory
dataset_path = '/content/tomatoleaf/'  # Adjust based on your dataset

# List files in the "tomato" directory
tomato_directory_path = os.path.join(dataset_path, 'tomato')
files_in_tomato_directory = os.listdir(tomato_directory_path)

# Print the list of files in the "tomato" directory
print("Files in the 'tomato' directory:", files_in_tomato_directory)
```

```
Files in the 'tomato' directory: ['val', 'cnn_train.py', 'train']
```

```
import os

# Specify the path to the dataset directory
dataset_path = '/content/tomatoleaf/'  # Adjust based on your dataset

# List files in the 'tomato/val' directory
val_directory_path = os.path.join(dataset_path, 'tomato', 'val')
files_in_val_directory = os.listdir(val_directory_path)

# List files in the 'tomato/train' directory
train_directory_path = os.path.join(dataset_path, 'tomato', 'train')
files_in_train_directory = os.listdir(train_directory_path)

# Print the lists of files in the 'val' and 'train' directories
print("Files in the 'val' directory:", files_in_val_directory)
print("Files in the 'train' directory:", files_in_train_directory)
```

```
Files in the 'val' directory: ['Tomato___Late_blight',
'Tomato___Tomato_mosaic_virus', 'Tomato___Leaf_Mold',
'Tomato___Spider_mites Two-spotted_spider_mite',
'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
'Tomato___Septoria_leaf_spot', 'Tomato___healthy',
'Tomato___Early_blight', 'Tomato___Bacterial_spot',
'Tomato___Target_Spot']
Files in the 'train' directory: ['Tomato___Late_blight',
'Tomato___Tomato_mosaic_virus', 'Tomato___Leaf_Mold',
'Tomato___Spider_mites Two-spotted_spider_mite',
'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
'Tomato___Septoria_leaf_spot', 'Tomato___healthy',
'Tomato___Early_blight', 'Tomato___Bacterial_spot',
'Tomato___Target_Spot']
```

```
# Specify the path to the 'train' directory
train_dir = '/content/tomatoleaf/tomato/train'
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    train_dir,
    seed=123,
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```

```
Found 10000 files belonging to 10 classes.
```

```
# Examine classname
class_names = dataset.class_names
class_names
```

```
['Tomato___Bacterial_spot',
 'Tomato___Early_blight',
 'Tomato___Late_blight',
 'Tomato___Leaf_Mold',
 'Tomato___Septoria_leaf_spot',
 'Tomato___Spider_mites Two-spotted_spider_mite',
 'Tomato___Target_Spot',
 'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
 'Tomato___Tomato_mosaic_virus',
 'Tomato___healthy']
```

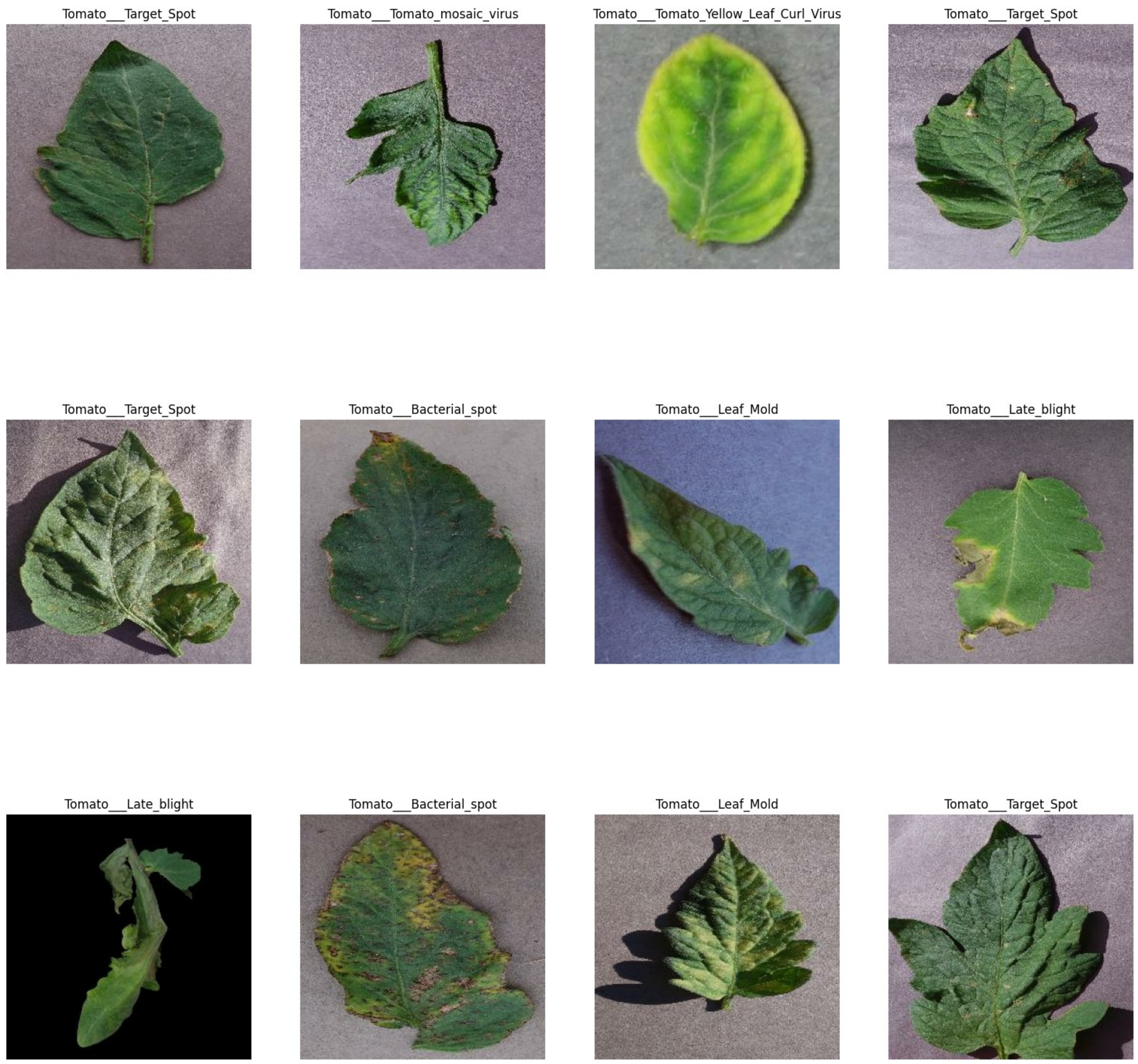
```
#Exploration of data
for image_batch, labels_batch in dataset.take(2):
    print(image_batch.shape)
    print(labels_batch.numpy())
```

```
(32, 256, 256, 3)
[5 0 0 2 3 6 6 4 6 4 0 3 4 0 1 5 7 5 2 8 3 6 8 3 2 4 0 7 8 6 1 5]
(32, 256, 256, 3)
[5 2 2 6 2 4 9 3 5 0 5 8 3 3 3 3 6 6 0 6 0 6 0 3 2 5 4 8 5 2 3 0]
```

So the data has been divided into 32 batchs.The code is for printing class of 32 images in each batches.The size of images is 256-256 pixels

Visualize some of the images from our dataset

```
plt.figure(figsize=(20, 20))
for image_batch, labels_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```



The data is visualized for the batch number 1 with the according name of disease!

How to divided data sources for modelling

Dataset should be divided into 3 subsets which are:

1. Training: Dataset to be used while training
2. Validation: Dataset to be tested against while training to help preventing overfitting and bias
3. Test: Dataset to be tested against after we trained a model

```
len(dataset)# is that because data has been fetched into batch : 
313*32 approximatelly 10000
```

```
313
```

```
train_size = 0.6 # 80% data is for trainning
len(dataset)*train_size
```

```
187.79999999999998
```

```
train_ds = dataset.take(250)# take only 20 first elements of dataset 
for testing and debuf method.
len(train_ds)
```

```
250
```

```
test_ds = dataset.skip(250)
len(test_ds)
```

```
63
```

```
val_size=0.1
len(dataset)*val_size
```

```
31.3
```

```
val_ds = test_ds.take(31)
len(val_ds)
```

```
31
```

```
test_ds = test_ds.skip(31)
len(test_ds)
```

```
32
```

```
#Creating function to divide dataset int train,validation and test set
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, 
test_split=0.1, shuffle=True, shuffle_size=10000):
    assert (train_split + test_split + val_split) == 1
    
    ds_size = len(ds)
    
    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=123)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds

train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)

len(train_ds)

250

len(val_ds)

31

len(test_ds)

32
```

Data processing when creating train,test dataset : Cache, Shuffle, and Prefetch the Dataset

###Cache:

Purpose: Caches elements of the dataset in memory, improving data loading speed. Why: Reduces the time spent reading data from disk by storing elements in memory after they are loaded for the first time. How: After an element is loaded from the dataset, it is cached in memory. Subsequent iterations can then use the cached data instead of reloading from the original data source.

###Shuffle: Purpose: Randomizes the order of examples in the dataset. Why: Prevents the model from learning patterns based on the order of the data How: The dataset is shuffled so that the model sees the examples in a different order.

###Prefetch: Purpose: Overlaps data loading and model execution. Why: Reduces training time How: During training, while the model is processing one batch, the input pipeline is preparing the next batch in parallel.

```
train_ds =
train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds =
val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds =
test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

The size of the whole dataset is 10000 so that we chose the number of shuffle is 1000

Classification Model

Data Processing before fectching into Neural Network:

First step: Resizing Images:adjust t o a consistent size that is suitable for model,ensuring that all images has same dimensions.In this case,set to 256*256 Steps 2: Normalization by divided by 256 so the values are in range (0,1). Images have pixal value in range(0,255).This helps to in stabilizing and accelerating the training process, avoiding numerical instability. Steps 3: Handling Predictions in Inference:

The model should be able to handle images of different sizes during inference even all the original data in dataset are already at size (256,256).The resizing layer becomes useful at inference time when someone supplies an image with dimensions other than (256, 256). The layer will resize the input image to the expected size before passing it through the rest of the model.

```
resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.experimental.preprocessing.Rescaling(1./255),
])
```

Data Augmentation

Why? Data Augmentation is generated for enhancing the accuracy of model. Besides, it is Better at Handling of Invariances. Data augmentation allows the model to learn these invariances by exposing it to transformed versions of the input data.

```
# we try with rotation and flipping first.
#data_augmentation = tf.keras.Sequential([

#layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical
"),
  #layers.experimental.preprocessing.RandomRotation(0.2),
#])

# Using 5 commons techniques
data_augmentation = tf.keras.Sequential([

layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"
),
  layers.experimental.preprocessing.RandomRotation(0.4),
  layers.experimental.preprocessing.RandomZoom(0.4),
  layers.experimental.preprocessing.RandomContrast(0.4),

layers.experimental.preprocessing.RandomTranslation(height_factor=0.1,
width_factor=0.1),
])
```

Applying Data Augmentation to Train Dataset

```
train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Model Architecture

CNN has been used. The initial layers is for resizing, normalization and data augmentation as said above. The final layers will be flattened and be applied Softmax activation function.

```
# Limit GPU memory usage
gpus = tf.config.experimental.list_physical_devices('GPU')

if gpus:
    try:
        tf.config.experimental.set_virtual_device_configuration(
            gpus[0],

[tf.config.experimental.VirtualDeviceConfiguration(memory_limit=1024)]
)
    except RuntimeError as e:
        print(e)
```

```
Virtual devices cannot be modified after being initialized
```

```
from tensorflow.keras.mixed_precision import Policy, set_global_policy

# Set mixed precision policy
policy = Policy('mixed_float16')
set_global_policy(policy)

from tensorflow.keras.mixed_precision import Policy, set_global_policy
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Set mixed precision policy
policy = Policy('mixed_float16')
set_global_policy(policy)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout,
Flatten, BatchNormalization

# Building model 1:
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 10

initializer = tf.keras.initializers.GlorotUniform(seed=42)
from tensorflow.keras import models, layers
```

```
model_1 = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu',
input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])

model_1.build(input_shape=input_shape)

# Building model 2
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 10
from tensorflow.keras import regularizers
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout ,
BatchNormalization
# In this code, We tried to add drop out with the ratio of 0.3
from tensorflow.keras import layers, models
model_2 = models.Sequential([
    resize_and_rescale,

    # Convolutional layers with padding and batch normalization
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=input_shape, padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, kernel_size=(3, 3), activation='relu',
padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, kernel_size=(3, 3), activation='relu',
padding='valid'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu',
padding='valid'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, kernel_size=(3, 3), activation='relu',
padding='valid'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, kernel_size=(3, 3), activation='relu',
padding='valid'),
    layers.BatchNormalization(),

    # Global Average Pooling layer
    layers.GlobalAveragePooling2D(),

    # Dense layers with dropout and L2 regularization
    layers.Dense(64, activation='relu',
kernel_regularizer=regularizers.l2(0.01)),
    layers.Dropout(0.3),
    layers.Dense(n_classes, activation='softmax',
kernel_regularizer=regularizers.l2(0.01)),
])

model_2.build(input_shape=input_shape)


model_1.summary()

Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0         

 conv2d_24 (Conv2D)          (32, 254, 254, 32)        896       

 max_pooling2d_23 (MaxPooli  (32, 127, 127, 32)        0         
 ng2D)                                                           

 conv2d_25 (Conv2D)          (32, 125, 125, 64)        18496     

 max_pooling2d_24 (MaxPooli  (32, 62, 62, 64)          0         
 ng2D)                                                           

 conv2d_26 (Conv2D)          (32, 60, 60, 64)          36928     

 max_pooling2d_25 (MaxPooli  (32, 30, 30, 64)          0         
 ng2D)
 conv2d_27 (Conv2D)          (32, 28, 28, 64)          36928

 max_pooling2d_26 (MaxPooli  (32, 14, 14, 64)          0
 ng2D)

 conv2d_28 (Conv2D)          (32, 12, 12, 64)          36928

 max_pooling2d_27 (MaxPooli  (32, 6, 6, 64)            0
 ng2D)

 conv2d_29 (Conv2D)          (32, 4, 4, 64)            36928

 max_pooling2d_28 (MaxPooli  (32, 2, 2, 64)            0
 ng2D)

 flatten_3 (Flatten)         (32, 256)                 0

 dense_8 (Dense)             (32, 64)                  16448

 dense_9 (Dense)             (32, 10)                  650

=================================================================
Total params: 184202 (719.54 KB)
Trainable params: 184202 (719.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model_2.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0

 conv2d_6 (Conv2D)           (32, 256, 256, 32)        896

 batch_normalization (Batch  (32, 256, 256, 32)        128
 Normalization)

 max_pooling2d_6 (MaxPoolin  (32, 128, 128, 32)        0
 g2D)

 conv2d_7 (Conv2D)           (32, 128, 128, 64)        18496

 batch_normalization_1 (Bat  (32, 128, 128, 64)        256
 chNormalization)

 max_pooling2d_7 (MaxPoolin  (32, 64, 64, 64)          0
 g2D)

 conv2d_8 (Conv2D)           (32, 62, 62, 64)          36928

 batch_normalization_2 (Bat  (32, 62, 62, 64)          256
 chNormalization)

 max_pooling2d_8 (MaxPoolin  (32, 31, 31, 64)          0
 g2D)

 conv2d_9 (Conv2D)           (32, 29, 29, 64)          36928

 batch_normalization_3 (Bat  (32, 29, 29, 64)          256
 chNormalization)

 max_pooling2d_9 (MaxPoolin  (32, 14, 14, 64)          0
 g2D)

 conv2d_10 (Conv2D)          (32, 12, 12, 64)          36928

 batch_normalization_4 (Bat  (32, 12, 12, 64)          256
 chNormalization)

 max_pooling2d_10 (MaxPooli  (32, 6, 6, 64)            0
 ng2D)

 conv2d_11 (Conv2D)          (32, 4, 4, 64)            36928

 batch_normalization_5 (Bat  (32, 4, 4, 64)            256
 chNormalization)

 global_average_pooling2d (  (32, 64)                  0
 GlobalAveragePooling2D)

 dense_2 (Dense)             (32, 64)                  4160

 dropout (Dropout)           (32, 64)                  0

 dense_3 (Dense)             (32, 10)                  650

=================================================================
Total params: 173322 (677.04 KB)
Trainable params: 172618 (674.29 KB)
Non-trainable params: 704 (2.75 KB)
_________________________________________________________________
```

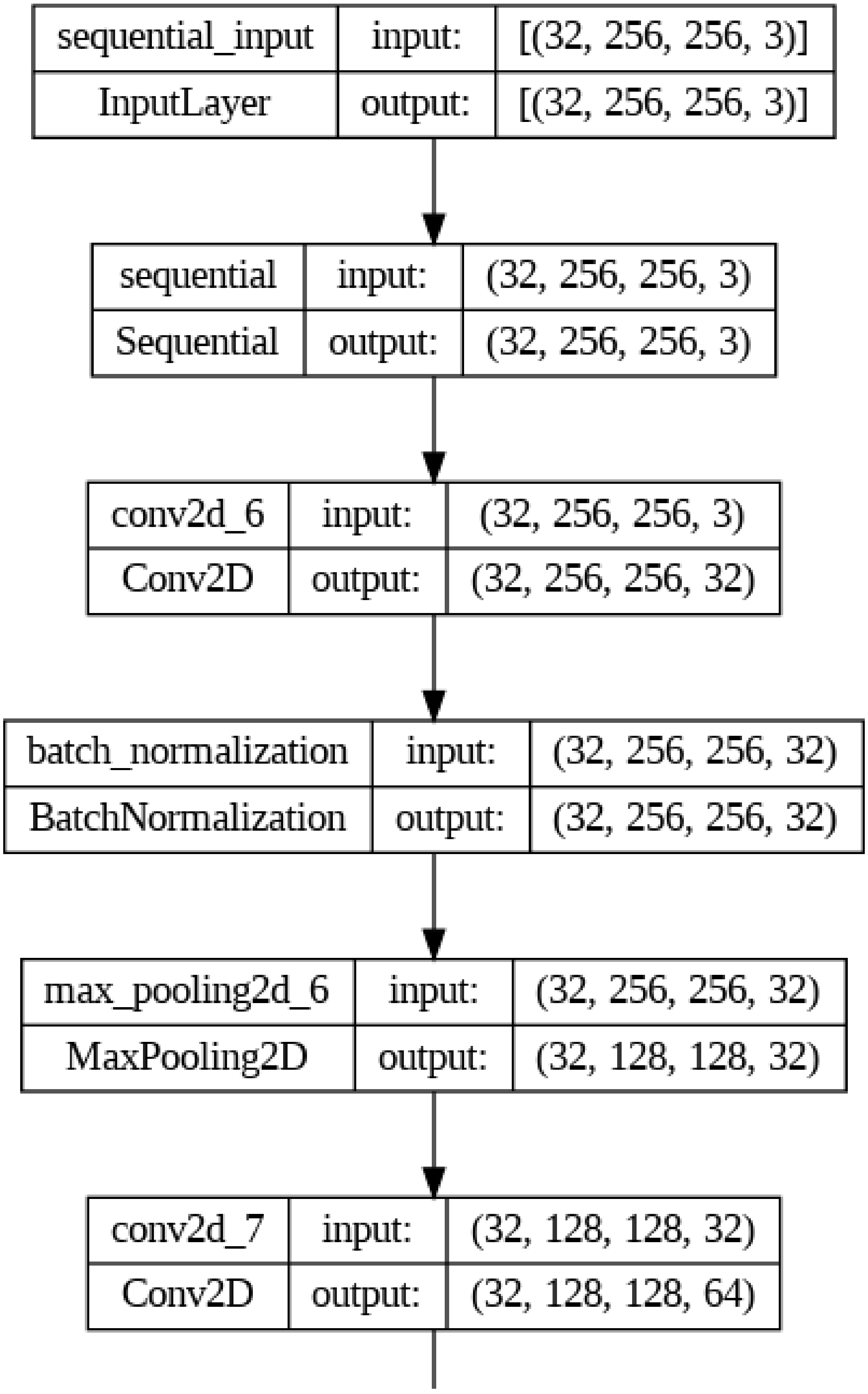
Compiling the Model

Optimizer function : Adam

`SparseCategoricalCrossentropy` for losses

`accuracy` as a metric for accesing accuracy and model performance

```
model_1.compile(
    optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)

#model_2.compile(
    #optimizer='adam',

#loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False)
,
    #metrics=['accuracy']
#)

# Compile model 2 using learing rate adjustment
from tensorflow.keras.optimizers import schedules

initial_learning_rate = 0.001
lr_schedule = schedules.ExponentialDecay(
    initial_learning_rate, decay_steps=10000, decay_rate=0.9,
staircase=True
)
optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)
model_2.compile(
    optimizer=optimizer,

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)

# Installation of Early Stopping
from keras.callbacks import EarlyStopping
calls = EarlyStopping(monitor = 'val_accuracy',
                      mode = 'max',
                      patience = 20,# stop if not improve for 20
consecutive epochs
                      verbose = 1,
                      restore_best_weights = True)# model weights are
reverted to the ones that gave the best monitored quantity value
during training.



# Structural visulization of Model 2
from tensorflow.keras.utils import plot_model
plot_model(model_2, show_shapes=True, to_file='model_1_plot.png')
```



sequential_input | InputLayer | input: | output: | [(32, 256, 256, 3)] | [(32, 256, 256, 3)]
sequential | Sequential | input: | output: | (32, 256, 256, 3) | (32, 256, 256, 3)
conv2d_6 | Conv2D | input: | output: | (32, 256, 256, 3) | (32, 256, 256, 32)
batch_normalization | BatchNormalization | input: | output: | (32, 256, 256, 32) | (32, 256, 256, 32)
max_pooling2d_6 | MaxPooling2D | input: | output: | (32, 256, 256, 32) | (32, 128, 128, 32)
conv2d_7 | Conv2D | input: | output: | (32, 128, 128, 32) | (32, 128, 128, 64)

```
# Calling and storing accuracy and loss of model 1
history_1 = model_1.fit(
   train_ds,
   batch_size=BATCH_SIZE,
   validation_data=val_ds,
   verbose=1,
   epochs=50,
)

Epoch 1/50
250/250 [==============================] - 185s 668ms/step - loss:
2.1901 - accuracy: 0.1766 - val_loss: 2.0146 - val_accuracy: 0.2369
Epoch 2/50
250/250 [==============================] - 164s 654ms/step - loss:
1.9382 - accuracy: 0.2882 - val_loss: 2.0467 - val_accuracy: 0.2722
Epoch 3/50
250/250 [==============================] - 164s 657ms/step - loss:
1.8137 - accuracy: 0.3442 - val_loss: 1.9328 - val_accuracy: 0.2762
Epoch 4/50
250/250 [==============================] - 162s 648ms/step - loss:
1.6272 - accuracy: 0.4163 - val_loss: 1.6853 - val_accuracy: 0.3921
Epoch 5/50
250/250 [==============================] - 161s 643ms/step - loss:
1.3412 - accuracy: 0.5252 - val_loss: 1.4953 - val_accuracy: 0.4960
Epoch 6/50
250/250 [==============================] - 161s 643ms/step - loss:
1.1632 - accuracy: 0.5859 - val_loss: 1.1370 - val_accuracy: 0.6119
Epoch 7/50
250/250 [==============================] - 162s 650ms/step - loss:
1.0340 - accuracy: 0.6349 - val_loss: 1.5148 - val_accuracy: 0.5716
Epoch 8/50
250/250 [==============================] - 160s 640ms/step - loss:
0.8986 - accuracy: 0.6807 - val_loss: 1.0022 - val_accuracy: 0.6704
Epoch 9/50
250/250 [==============================] - 160s 638ms/step - loss:
0.8065 - accuracy: 0.7144 - val_loss: 1.0595 - val_accuracy: 0.6804
Epoch 10/50
250/250 [==============================] - 159s 634ms/step - loss:
0.7490 - accuracy: 0.7353 - val_loss: 3.0727 - val_accuracy: 0.4345
Epoch 11/50
250/250 [==============================] - 160s 640ms/step - loss:
0.7160 - accuracy: 0.7472 - val_loss: 1.5683 - val_accuracy: 0.5877
Epoch 12/50
250/250 [==============================] - 159s 635ms/step - loss:
0.6441 - accuracy: 0.7705 - val_loss: 1.0290 - val_accuracy: 0.7127
Epoch 13/50
250/250 [==============================] - 160s 639ms/step - loss:
0.6048 - accuracy: 0.7849 - val_loss: 0.9387 - val_accuracy: 0.7228
Epoch 14/50
250/250 [==============================] - 159s 637ms/step - loss:
```

```
0.5814 - accuracy: 0.7985 - val_loss: 2.0463 - val_accuracy: 0.5817
Epoch 15/50
250/250 [==============================] - 160s 638ms/step - loss:
0.5610 - accuracy: 0.8006 - val_loss: 1.9839 - val_accuracy: 0.5877
Epoch 16/50
250/250 [==============================] - 159s 636ms/step - loss:
0.5207 - accuracy: 0.8193 - val_loss: 2.1346 - val_accuracy: 0.5867
Epoch 17/50
250/250 [==============================] - 159s 634ms/step - loss:
0.5125 - accuracy: 0.8224 - val_loss: 0.6996 - val_accuracy: 0.7772
Epoch 18/50
250/250 [==============================] - 159s 634ms/step - loss:
0.4563 - accuracy: 0.8383 - val_loss: 1.0665 - val_accuracy: 0.7067
Epoch 19/50
250/250 [==============================] - 159s 637ms/step - loss:
0.4584 - accuracy: 0.8384 - val_loss: 2.3154 - val_accuracy: 0.5746
Epoch 20/50
250/250 [==============================] - 159s 636ms/step - loss:
0.4184 - accuracy: 0.8502 - val_loss: 0.8582 - val_accuracy: 0.7581
Epoch 21/50
250/250 [==============================] - 159s 637ms/step - loss:
0.3891 - accuracy: 0.8659 - val_loss: 0.9607 - val_accuracy: 0.7399
Epoch 22/50
250/250 [==============================] - 158s 633ms/step - loss:
0.4071 - accuracy: 0.8587 - val_loss: 1.4069 - val_accuracy: 0.6825
Epoch 23/50
250/250 [==============================] - 158s 632ms/step - loss:
0.4059 - accuracy: 0.8573 - val_loss: 0.8352 - val_accuracy: 0.7772
Epoch 24/50
250/250 [==============================] - 158s 631ms/step - loss:
0.3548 - accuracy: 0.8729 - val_loss: 0.8225 - val_accuracy: 0.7722
Epoch 25/50
250/250 [==============================] - 159s 635ms/step - loss:
0.3748 - accuracy: 0.8694 - val_loss: 0.5074 - val_accuracy: 0.8458
Epoch 26/50
250/250 [==============================] - 158s 632ms/step - loss:
0.3654 - accuracy: 0.8682 - val_loss: 0.7380 - val_accuracy: 0.8044
Epoch 27/50
250/250 [==============================] - 158s 632ms/step - loss:
0.3405 - accuracy: 0.8829 - val_loss: 0.3761 - val_accuracy: 0.8800
Epoch 28/50
250/250 [==============================] - 158s 630ms/step - loss:
0.3313 - accuracy: 0.8843 - val_loss: 0.6159 - val_accuracy: 0.8216
Epoch 29/50
250/250 [==============================] - 157s 629ms/step - loss:
0.3291 - accuracy: 0.8865 - val_loss: 0.9423 - val_accuracy: 0.7833
Epoch 30/50
250/250 [==============================] - 158s 632ms/step - loss:
0.3110 - accuracy: 0.8920 - val_loss: 0.3735 - val_accuracy: 0.8800
```

```
Epoch 31/50
250/250 [==============================] - 159s 636ms/step - loss:
0.2986 - accuracy: 0.8958 - val_loss: 0.9200 - val_accuracy: 0.7671
Epoch 32/50
250/250 [==============================] - 159s 635ms/step - loss:
0.2793 - accuracy: 0.9019 - val_loss: 0.8130 - val_accuracy: 0.7833
Epoch 33/50
250/250 [==============================] - 160s 640ms/step - loss:
0.2985 - accuracy: 0.8950 - val_loss: 1.1669 - val_accuracy: 0.7601
Epoch 34/50
250/250 [==============================] - 159s 635ms/step - loss:
0.3158 - accuracy: 0.8902 - val_loss: 2.7496 - val_accuracy: 0.5958
Epoch 35/50
250/250 [==============================] - 160s 640ms/step - loss:
0.2764 - accuracy: 0.9062 - val_loss: 1.3402 - val_accuracy: 0.7490
Epoch 36/50
250/250 [==============================] - 159s 636ms/step - loss:
0.2977 - accuracy: 0.8973 - val_loss: 1.1514 - val_accuracy: 0.7288
Epoch 37/50
250/250 [==============================] - 159s 637ms/step - loss:
0.2723 - accuracy: 0.9033 - val_loss: 1.1122 - val_accuracy: 0.7409
Epoch 38/50
250/250 [==============================] - 158s 634ms/step - loss:
0.2797 - accuracy: 0.9034 - val_loss: 0.3767 - val_accuracy: 0.8921
Epoch 39/50
250/250 [==============================] - 159s 634ms/step - loss:
0.2640 - accuracy: 0.9103 - val_loss: 0.4383 - val_accuracy: 0.8609
Epoch 40/50
250/250 [==============================] - 158s 630ms/step - loss:
0.2793 - accuracy: 0.9036 - val_loss: 0.9367 - val_accuracy: 0.7792
Epoch 41/50
250/250 [==============================] - 159s 637ms/step - loss:
0.2522 - accuracy: 0.9124 - val_loss: 0.6718 - val_accuracy: 0.8337
Epoch 42/50
250/250 [==============================] - 159s 635ms/step - loss:
0.2747 - accuracy: 0.9067 - val_loss: 1.2926 - val_accuracy: 0.7288
Epoch 43/50
250/250 [==============================] - 161s 644ms/step - loss:
0.2399 - accuracy: 0.9130 - val_loss: 0.6323 - val_accuracy: 0.8407
Epoch 44/50
250/250 [==============================] - 160s 641ms/step - loss:
0.2397 - accuracy: 0.9138 - val_loss: 0.3574 - val_accuracy: 0.8921
Epoch 45/50
250/250 [==============================] - 159s 637ms/step - loss:
0.2420 - accuracy: 0.9193 - val_loss: 0.5615 - val_accuracy: 0.8538
Epoch 46/50
250/250 [==============================] - 159s 637ms/step - loss:
0.2380 - accuracy: 0.9188 - val_loss: 0.6123 - val_accuracy: 0.8135
Epoch 47/50
```

```
250/250 [==============================] - 159s 638ms/step - loss:
0.2173 - accuracy: 0.9238 - val_loss: 0.9168 - val_accuracy: 0.7913
Epoch 48/50
250/250 [==============================] - 158s 632ms/step - loss:
0.2271 - accuracy: 0.9212 - val_loss: 0.5871 - val_accuracy: 0.8579
Epoch 49/50
250/250 [==============================] - 159s 635ms/step - loss:
0.2161 - accuracy: 0.9250 - val_loss: 0.5872 - val_accuracy: 0.8508
Epoch 50/50
250/250 [==============================] - 159s 636ms/step - loss:
0.2053 - accuracy: 0.9280 - val_loss: 0.2669 - val_accuracy: 0.9123

# Calling and storing accuracy and loss of model 2
history_2= model_2.fit(train_ds,
                    callbacks = [calls],
                    steps_per_epoch = 8000/32,
                    epochs = 50,
                    batch_size=BATCH_SIZE,
                    validation_steps = 1000/32,
                    validation_data = val_ds)

Epoch 1/50
250/250 [==============================] - 170s 680ms/step - loss:
1.0759 - accuracy: 0.7417 - val_loss: 1.9240 - val_accuracy: 0.4567
Epoch 2/50
250/250 [==============================] - 169s 676ms/step - loss:
0.9230 - accuracy: 0.7779 - val_loss: 2.0948 - val_accuracy: 0.4869
Epoch 3/50
250/250 [==============================] - 169s 675ms/step - loss:
0.8389 - accuracy: 0.7983 - val_loss: 2.0391 - val_accuracy: 0.5091
Epoch 4/50
250/250 [==============================] - 168s 670ms/step - loss:
0.7537 - accuracy: 0.8210 - val_loss: 1.5080 - val_accuracy: 0.6058
Epoch 5/50
250/250 [==============================] - 168s 671ms/step - loss:
0.6968 - accuracy: 0.8352 - val_loss: 1.3083 - val_accuracy: 0.6361
Epoch 6/50
250/250 [==============================] - 167s 668ms/step - loss:
0.6355 - accuracy: 0.8561 - val_loss: 1.0506 - val_accuracy: 0.6905
Epoch 7/50
250/250 [==============================] - 168s 670ms/step - loss:
0.5873 - accuracy: 0.8704 - val_loss: 2.9079 - val_accuracy: 0.4677
Epoch 8/50
250/250 [==============================] - 166s 665ms/step - loss:
0.5698 - accuracy: 0.8731 - val_loss: 1.2961 - val_accuracy: 0.6794
Epoch 9/50
250/250 [==============================] - 166s 662ms/step - loss:
0.5404 - accuracy: 0.8810 - val_loss: 1.1912 - val_accuracy: 0.6704
Epoch 10/50
250/250 [==============================] - 167s 668ms/step - loss:
```

```
0.5088 - accuracy: 0.8924 - val_loss: 0.8487 - val_accuracy: 0.7671
Epoch 11/50
250/250 [==============================] - 166s 666ms/step - loss:
0.4788 - accuracy: 0.9004 - val_loss: 2.1982 - val_accuracy: 0.5312
Epoch 12/50
250/250 [==============================] - 167s 668ms/step - loss:
0.4667 - accuracy: 0.9089 - val_loss: 0.8465 - val_accuracy: 0.7500
Epoch 13/50
250/250 [==============================] - 167s 667ms/step - loss:
0.4638 - accuracy: 0.9044 - val_loss: 1.7372 - val_accuracy: 0.5806
Epoch 14/50
250/250 [==============================] - 167s 668ms/step - loss:
0.4515 - accuracy: 0.9043 - val_loss: 0.7315 - val_accuracy: 0.8175
Epoch 15/50
250/250 [==============================] - 167s 670ms/step - loss:
0.4281 - accuracy: 0.9113 - val_loss: 0.5955 - val_accuracy: 0.8518
Epoch 16/50
250/250 [==============================] - 168s 670ms/step - loss:
0.4049 - accuracy: 0.9193 - val_loss: 1.1320 - val_accuracy: 0.7067
Epoch 17/50
250/250 [==============================] - 166s 664ms/step - loss:
0.4089 - accuracy: 0.9167 - val_loss: 0.7619 - val_accuracy: 0.7903
Epoch 18/50
250/250 [==============================] - 166s 662ms/step - loss:
0.3894 - accuracy: 0.9235 - val_loss: 0.7557 - val_accuracy: 0.7984
Epoch 19/50
250/250 [==============================] - 166s 663ms/step - loss:
0.3704 - accuracy: 0.9300 - val_loss: 0.5130 - val_accuracy: 0.8740
Epoch 20/50
250/250 [==============================] - 166s 664ms/step - loss:
0.3688 - accuracy: 0.9290 - val_loss: 1.0150 - val_accuracy: 0.7389
Epoch 21/50
250/250 [==============================] - 166s 665ms/step - loss:
0.3747 - accuracy: 0.9262 - val_loss: 0.5056 - val_accuracy: 0.8780
Epoch 22/50
250/250 [==============================] - 165s 659ms/step - loss:
0.3605 - accuracy: 0.9332 - val_loss: 0.4646 - val_accuracy: 0.8942
Epoch 23/50
250/250 [==============================] - 165s 661ms/step - loss:
0.3506 - accuracy: 0.9351 - val_loss: 0.7402 - val_accuracy: 0.7974
Epoch 24/50
250/250 [==============================] - 166s 662ms/step - loss:
0.3428 - accuracy: 0.9361 - val_loss: 0.7275 - val_accuracy: 0.8024
Epoch 25/50
250/250 [==============================] - 167s 666ms/step - loss:
0.3333 - accuracy: 0.9362 - val_loss: 1.3959 - val_accuracy: 0.6744
Epoch 26/50
250/250 [==============================] - 166s 664ms/step - loss:
0.3289 - accuracy: 0.9386 - val_loss: 0.4452 - val_accuracy: 0.8921
```

```
Epoch 27/50
250/250 [==============================] - 167s 669ms/step - loss:
0.3099 - accuracy: 0.9413 - val_loss: 0.7816 - val_accuracy: 0.8085
Epoch 28/50
250/250 [==============================] - 166s 663ms/step - loss:
0.3097 - accuracy: 0.9466 - val_loss: 0.4113 - val_accuracy: 0.8962
Epoch 29/50
250/250 [==============================] - 166s 664ms/step - loss:
0.3212 - accuracy: 0.9366 - val_loss: 1.1827 - val_accuracy: 0.6694
Epoch 30/50
250/250 [==============================] - 165s 659ms/step - loss:
0.3125 - accuracy: 0.9420 - val_loss: 0.7742 - val_accuracy: 0.8075
Epoch 31/50
250/250 [==============================] - 167s 666ms/step - loss:
0.3043 - accuracy: 0.9435 - val_loss: 0.6486 - val_accuracy: 0.8256
Epoch 32/50
250/250 [==============================] - 166s 664ms/step - loss:
0.2886 - accuracy: 0.9463 - val_loss: 1.1466 - val_accuracy: 0.6925
Epoch 33/50
250/250 [==============================] - 166s 665ms/step - loss:
0.2935 - accuracy: 0.9464 - val_loss: 0.6661 - val_accuracy: 0.8065
Epoch 34/50
250/250 [==============================] - 165s 660ms/step - loss:
0.2899 - accuracy: 0.9458 - val_loss: 0.7506 - val_accuracy: 0.8256
Epoch 35/50
250/250 [==============================] - 166s 662ms/step - loss:
0.2909 - accuracy: 0.9448 - val_loss: 0.8537 - val_accuracy: 0.7792
Epoch 36/50
250/250 [==============================] - 166s 663ms/step - loss:
0.2913 - accuracy: 0.9459 - val_loss: 0.2530 - val_accuracy: 0.9556
Epoch 37/50
250/250 [==============================] - 165s 660ms/step - loss:
0.2500 - accuracy: 0.9582 - val_loss: 0.4489 - val_accuracy: 0.8851
Epoch 38/50
250/250 [==============================] - 165s 660ms/step - loss:
0.2700 - accuracy: 0.9519 - val_loss: 0.9754 - val_accuracy: 0.7611
Epoch 39/50
250/250 [==============================] - 166s 662ms/step - loss:
0.2521 - accuracy: 0.9573 - val_loss: 0.2396 - val_accuracy: 0.9556
Epoch 40/50
250/250 [==============================] - 165s 659ms/step - loss:
0.2492 - accuracy: 0.9578 - val_loss: 0.3404 - val_accuracy: 0.9264
Epoch 41/50
250/250 [==============================] - 166s 664ms/step - loss:
0.2537 - accuracy: 0.9544 - val_loss: 0.2864 - val_accuracy: 0.9355
Epoch 42/50
250/250 [==============================] - 165s 661ms/step - loss:
0.2475 - accuracy: 0.9578 - val_loss: 0.5719 - val_accuracy: 0.8387
Epoch 43/50
```

```
250/250 [==============================] - 165s 661ms/step - loss:
0.2451 - accuracy: 0.9587 - val_loss: 1.4211 - val_accuracy: 0.7087
Epoch 44/50
250/250 [==============================] - 164s 657ms/step - loss:
0.2398 - accuracy: 0.9579 - val_loss: 0.7685 - val_accuracy: 0.7933
Epoch 45/50
250/250 [==============================] - 164s 657ms/step - loss:
0.2376 - accuracy: 0.9564 - val_loss: 0.6409 - val_accuracy: 0.8337
Epoch 46/50
250/250 [==============================] - 164s 658ms/step - loss:
0.2412 - accuracy: 0.9568 - val_loss: 0.4385 - val_accuracy: 0.8901
Epoch 47/50
250/250 [==============================] - 165s 661ms/step - loss:
0.2450 - accuracy: 0.9537 - val_loss: 0.3232 - val_accuracy: 0.9294
Epoch 48/50
250/250 [==============================] - 164s 658ms/step - loss:
0.2235 - accuracy: 0.9622 - val_loss: 0.7746 - val_accuracy: 0.8196
Epoch 49/50
250/250 [==============================] - 165s 659ms/step - loss:
0.2366 - accuracy: 0.9582 - val_loss: 1.1750 - val_accuracy: 0.7389
Epoch 50/50
250/250 [==============================] - 166s 664ms/step - loss:
0.2259 - accuracy: 0.9607 - val_loss: 0.9216 - val_accuracy: 0.7853

print(val_ds)

<_PrefetchDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3),
dtype=tf.float32, name=None), TensorSpec(shape=(None,),
dtype=tf.int32, name=None))>

scores_1 = model_1.evaluate(test_ds)

32/32 [==============================] - 1s 16ms/step - loss: 0.1497 -
accuracy: 0.9512

scores_1# this is list containing of losses and accuracy value

[0.2321028709411621, 0.9326171875]
```

Plotting the Accuracy and Loss Curves

```
history_1

<keras.src.callbacks.History at 0x7aaf015ef100>

history_1.params

{'verbose': 1, 'epochs': 1, 'steps': 250}

history_1.history.keys()

dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

loss, accuracy, val loss etc are a python list containing values of loss, accuracy etc at the end of each epoch

```
type(history_1.history['loss'])
```

```
list
```

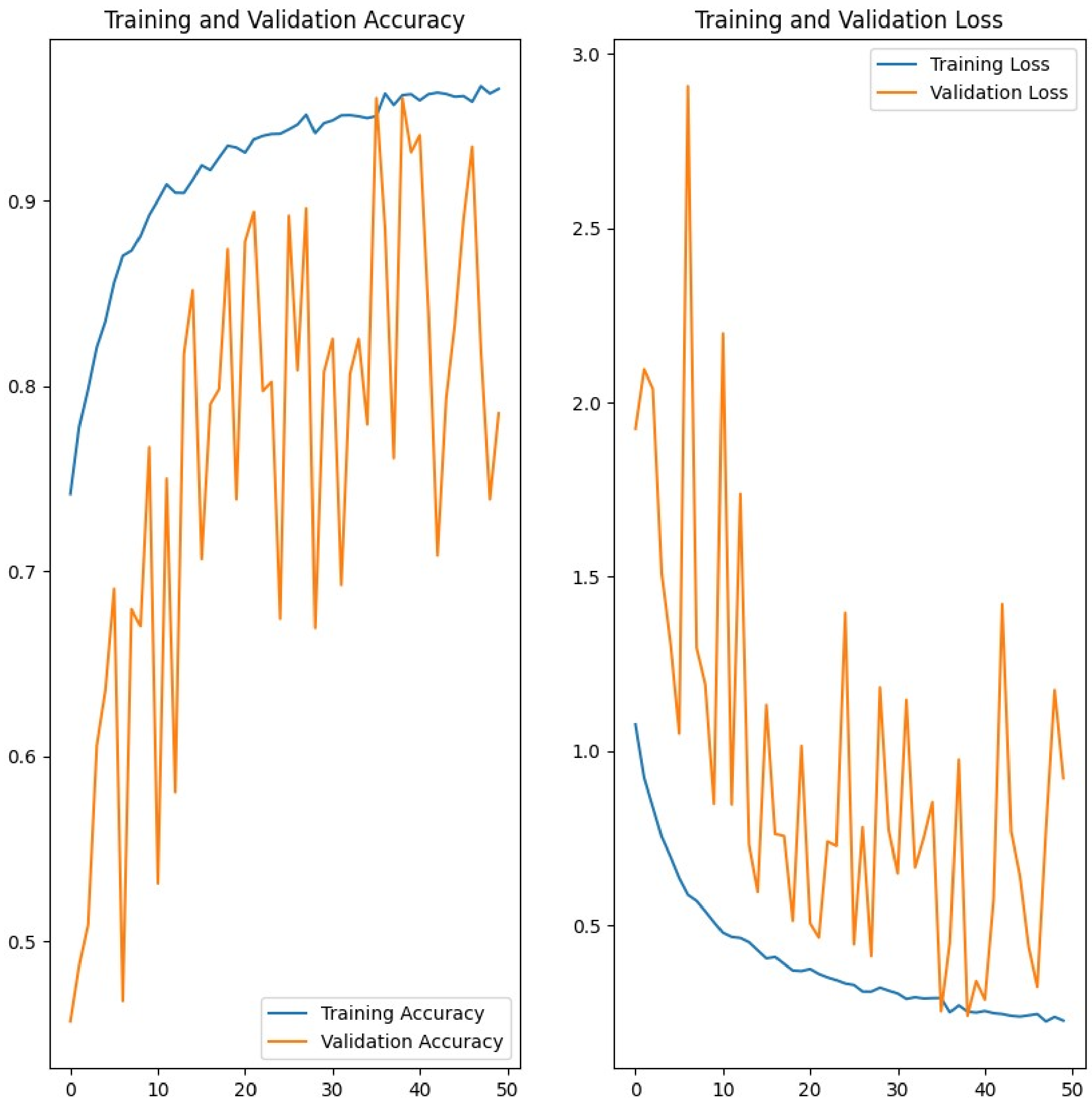
```
len(history_1.history['loss'])
```

```
50
```

```
history_1.history['loss'][:10] # show loss for first 10 epochs
```

```
[0.13088203966617584]
```

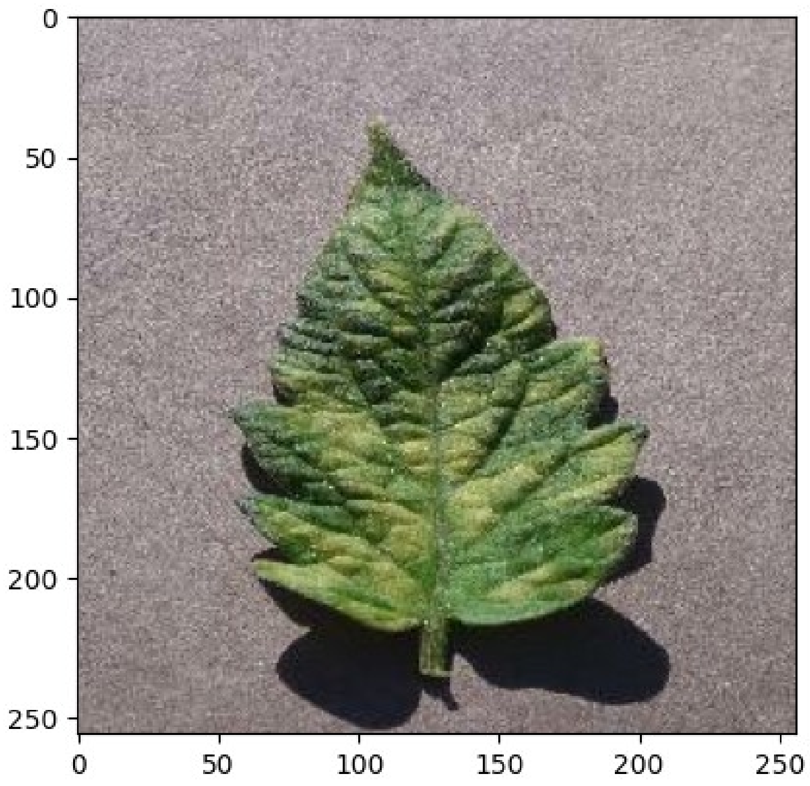
```
acc_1 = history_1.history['accuracy']
val_acc_1 = history_1.history['val_accuracy']

loss_1 = history_1.history['loss']
val_loss_1 = history_1.history['val_loss']

acc_2 = history_2.history['accuracy']
val_acc_2 = history_2.history['val_accuracy']

loss_2 = history_2.history['loss']
val_loss_2 = history_2.history['val_loss']

# Plotting of learning curve for model 1
EPOCHS = 50
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc_1, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc_1, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss_1, label='Training Loss')
plt.plot(range(EPOCHS), val_loss_1, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

# Plotting learning curve for model 2
plt.figure(figsize=(10, 10))
plt.subplot(1, 2, 1)
plt.plot(range(50), acc_2, label='Training Accuracy')
plt.plot(range(50), val_acc_2, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(50), loss_2, label='Training Loss')
plt.plot(range(50), val_loss_2, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



Run prediction on a sample image

```
import numpy as np
for images_batch, labels_batch in test_ds.take(1):

    first_image = images_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()

    print("first image to predict")
    plt.imshow(first_image)
    print("actual label:",class_names[first_label])

    batch_prediction_1 = model_1.predict(images_batch)
    print("predicted 
label:",class_names[np.argmax(batch_prediction_1[0])])
```

```
first image to predict
actual label: Tomato___Leaf_Mold
1/1 [==============================] - 0s 148ms/step
predicted label: Tomato___Spider_mites Two-spotted_spider_mite
```

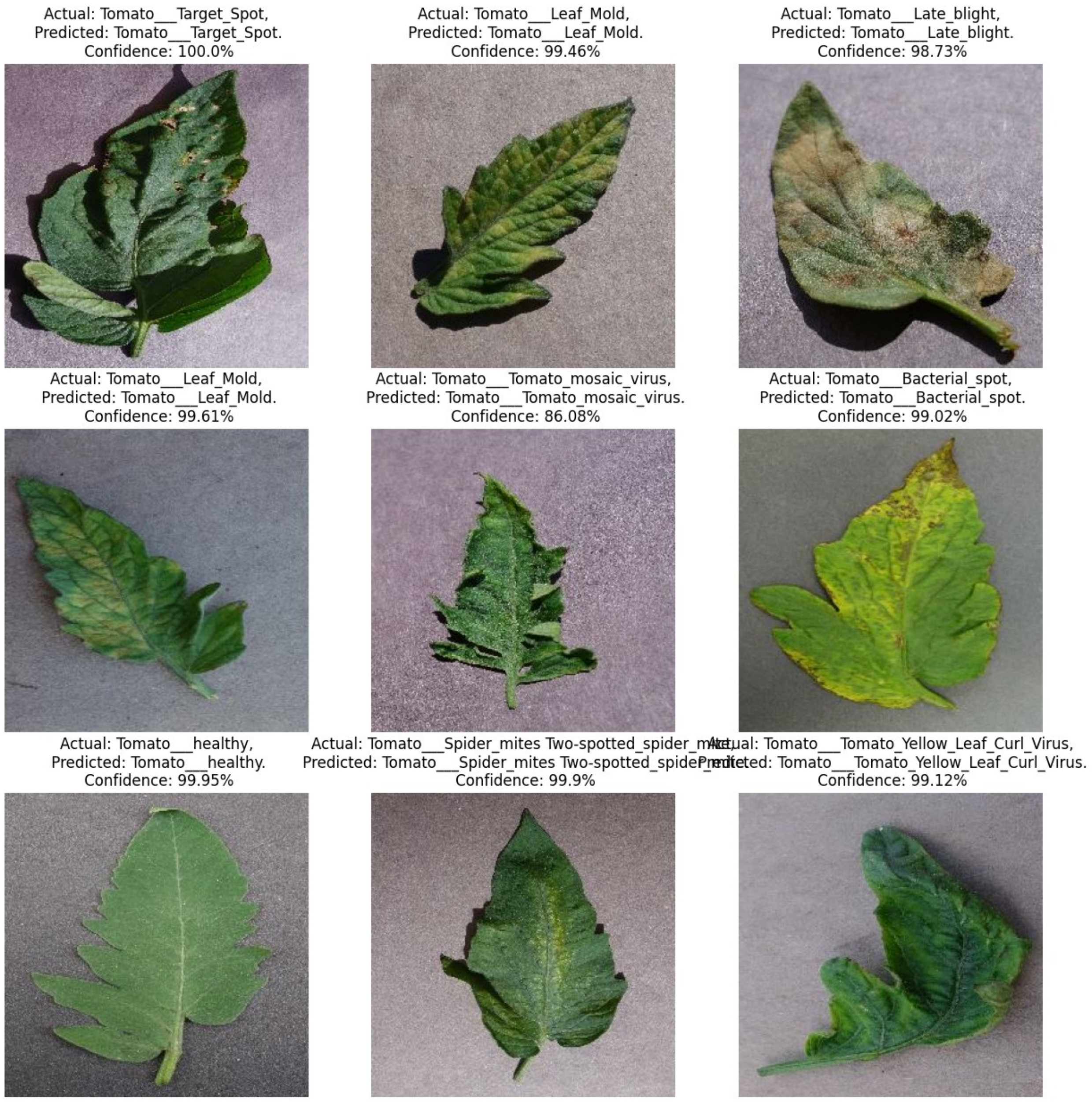


Write a function for inference

```
def predict(model, img):
    img_array = 
tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence
```

Now run inference on few sample images

```
plt.figure(figsize=(15, 15))
for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model_1, 
images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted: 
{predicted_class}.\n Confidence: {confidence}%")

        plt.axis("off")

1/1 [==============================] - 0s 381ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
```


Actual: Tomato___Target_Spot,
Predicted: Tomato___Target_Spot.
Confidence: 100.0%
Actual: Tomato___Leaf_Mold,
Predicted: Tomato___Leaf_Mold.
Confidence: 99.46%
Actual: Tomato___Late_blight,
Predicted: Tomato___Late_blight.
Confidence: 98.73%
Actual: Tomato___Leaf_Mold,
Predicted: Tomato___Leaf_Mold.
Confidence: 99.61%
Actual: Tomato___Tomato_mosaic_virus,
Predicted: Tomato___Tomato_mosaic_virus.
Confidence: 86.08%
Actual: Tomato___Bacterial_spot,
Predicted: Tomato___Bacterial_spot.
Confidence: 99.02%
Actual: Tomato___healthy,
Predicted: Tomato___healthy.
Confidence: 99.95%
Actual: Tomato___Spider_mites Two-spotted_spider_mite,
Predicted: Tomato___Spider_mites Two-spotted_spider_mite.
Confidence: 99.9%
Actual: Tomato___Tomato_Yellow_Leaf_Curl_Virus,
Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus.
Confidence: 99.12%

```python
# Model performance assessment using Confusion matrix, Preciosn, and recall
from sklearn.metrics import confusion_matrix, precision_score, recall_score
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
def predict_and_evaluate(model, test_ds, class_names):
    predicted_labels = []
    true_labels = []
    for images, labels in test_ds:
        for i in range(len(labels)):
            img_array =
tf.keras.preprocessing.image.img_to_array(images[i].numpy())
            img_array = tf.expand_dims(img_array, 0)

            predictions = model.predict(img_array)

            predicted_class = class_names[np.argmax(predictions[0])]
            confidence = np.max(predictions[0])
            predicted_labels.append(predicted_class)
            true_labels.append(class_names[labels[i]])

    # Calculate confusion matrix
    cm = confusion_matrix(true_labels, predicted_labels,
labels=class_names)
    # Calculate precision and recall scores
    precision = precision_score(true_labels, predicted_labels,
labels=class_names, average='weighted')
    recall = recall_score(true_labels, predicted_labels,
labels=class_names, average='weighted')

    # Print precision and recall scores
    print(f'Precision for model 2: {precision:.4f}')
    print(f'Recall for model 2: {recall:.4f}')
    # Plot confusion matrix
    plt.figure(figsize=(10, 10))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_names, yticklabels=class_names)
    plt.xlabel('Predicted disease')
    plt.ylabel('Actual disease')
    plt.title('Confusion Matrix for model 2 ')
    plt.show()
class_names = dataset.class_names
# Call the function with your model and test dataset
predict_and_evaluate(model_2, test_ds, class_names)
```

```
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 27ms/step
```

```
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
```

```
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
```

```
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
```

```
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
```

```
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
```

```
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
```

```
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
```

```
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
```

```
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
```

```
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
```

```
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
```

```
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
```

```
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
```

```
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
```

```
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
```

```
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
```

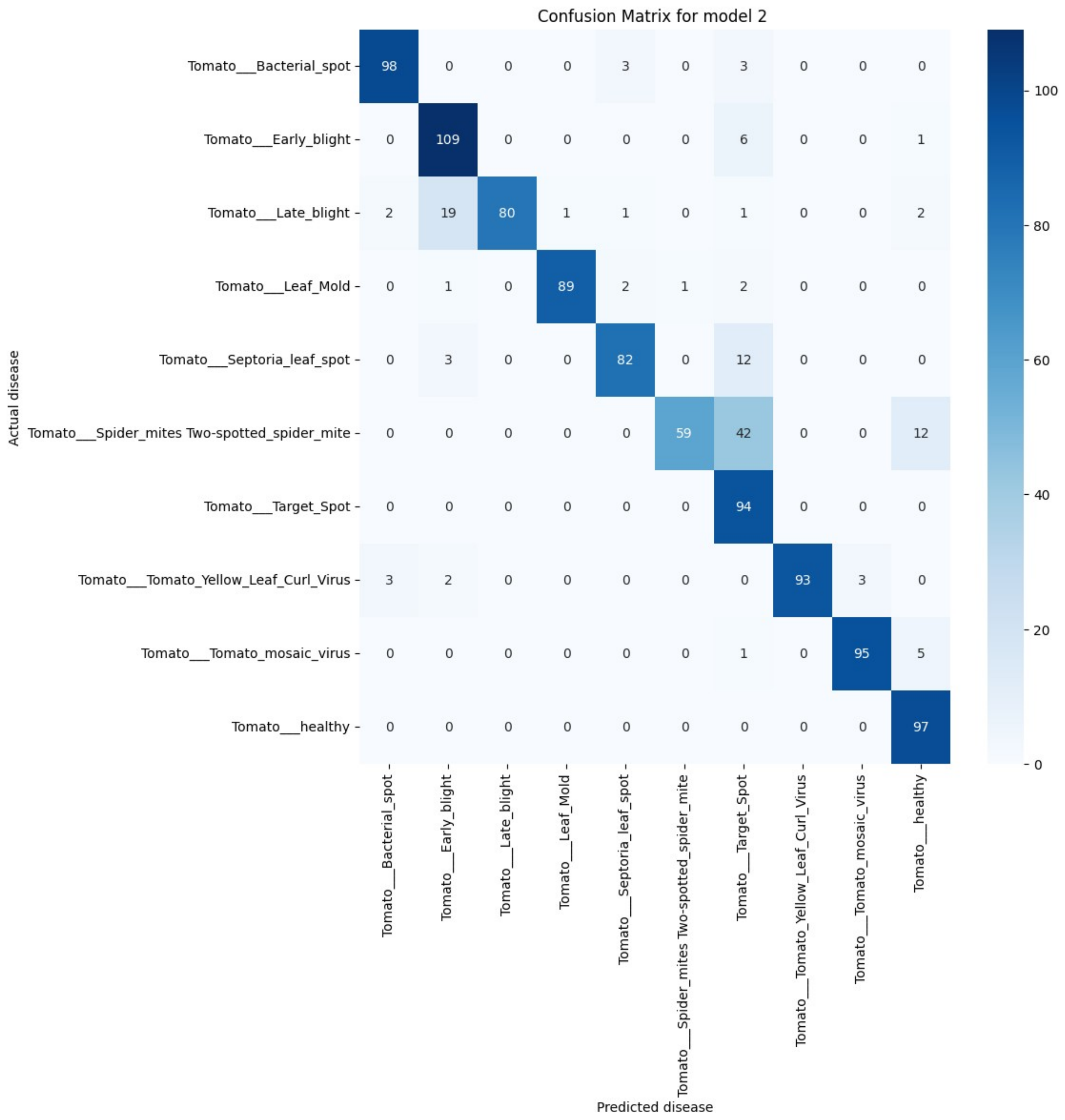
```
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
```

```
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
```

```
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
```

```
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
```

```
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
Precision for model 2: 0.9072
Recall for model 2: 0.8750
```



	Tomato___Bacterial_spot	Tomato___Early_blight	Tomato___Late_blight	Tomato___Leaf_Mold	Tomato___Septoria_leaf_spot	Tomato___Spider_mites Two-spotted_spider_mite	Tomato___Target_Spot	Tomato___Tomato_Yellow_Leaf_Curl_Virus	Tomato___Tomato_mosaic_virus	Tomato___healthy
Tomato___Bacterial_spot	98	0	0	0	3	0	3	0	0	0
Tomato___Early_blight	0	109	0	0	0	0	6	0	0	1
Tomato___Late_blight	2	19	80	1	1	0	1	0	0	2
Tomato___Leaf_Mold	0	1	0	89	2	1	2	0	0	0
Tomato___Septoria_leaf_spot	0	3	0	0	82	0	12	0	0	0
Tomato___Spider_mites Two-spotted_spider_mite	0	0	0	0	0	59	42	0	0	12
Tomato___Target_Spot	0	0	0	0	0	0	94	0	0	0
Tomato___Tomato_Yellow_Leaf_Curl_Virus	3	2	0	0	0	0	0	93	3	0
Tomato___Tomato_mosaic_virus	0	0	0	0	0	0	1	0	95	5
Tomato___healthy	0	0	0	0	0	0	0	0	0	97

```
#Test accurracy through 65 epochs for model 1
import numpy as np
import matplotlib.pyplot as plt
test_accuracy_history = []
epochs = 65
for epoch in range(epochs):
    history_1 = model_1.fit(
        train_ds,
        batch_size=BATCH_SIZE,
        validation_data=val_ds,
        verbose=1,
        epochs=1,
    )


    test_loss, test_accuracy = model_1.evaluate(test_ds)
    print(f'Epoch {epoch + 1}/{epochs} - Test Accuracy: {test_accuracy
* 100:.2f}%')

    test_accuracy_history.append(test_accuracy)

# Calculate the mean test accuracy for model 1
mean_test_accuracy_1 = np.mean(test_accuracy_history)
print(f'Mean Test Accuracy for model 1 over {epochs} epochs:
{mean_test_accuracy_1 * 100:.2f}%')
```

```
Mean Test Accuracy for model 1 over 65 epochs: 89.85%
```

```
#Test accuracy through 65 epochs of model 2
test_accuracy_history_2 = []
epochs = 65
for epoch in range(epochs):
    history_2 = model_2.fit(
        train_ds,
        batch_size=BATCH_SIZE,
        validation_data=val_ds,
        verbose=1,
        epochs=1,
    )


    test_loss_2, test_accuracy_2 = model_2.evaluate(test_ds)
    print(f'Epoch {epoch + 1}/{epochs} - Test Accuracy:
{test_accuracy_2 * 100:.2f}%')

    test_accuracy_history_2.append(test_accuracy_2)
```

```
250/250 [==============================] - 170s 681ms/step - loss:
0.2091 - accuracy: 0.9654 - val_loss: 0.3454 - val_accuracy: 0.9123
32/32 [==============================] - 1s 17ms/step - loss: 0.3690 -
accuracy: 0.9053
Epoch 1/65 - Test Accuracy: 90.53%
250/250 [==============================] - 169s 675ms/step - loss:
0.2074 - accuracy: 0.9658 - val_loss: 0.3707 - val_accuracy: 0.9083
32/32 [==============================] - 1s 17ms/step - loss: 0.3322 -
accuracy: 0.9170
Epoch 2/65 - Test Accuracy: 91.70%
250/250 [==============================] - 168s 672ms/step - loss:
0.2069 - accuracy: 0.9629 - val_loss: 0.4340 - val_accuracy: 0.8821
32/32 [==============================] - 1s 17ms/step - loss: 0.4456 -
accuracy: 0.8848
```

```
Epoch 3/65 - Test Accuracy: 88.48%
250/250 [==============================] - 169s 677ms/step - loss: 
0.2083 - accuracy: 0.9657 - val_loss: 0.4362 - val_accuracy: 0.8952
32/32 [==============================] - 1s 17ms/step - loss: 0.4260 - 
accuracy: 0.9033
Epoch 4/65 - Test Accuracy: 90.33%
250/250 [==============================] - 168s 673ms/step - loss: 
0.2070 - accuracy: 0.9656 - val_loss: 0.8146 - val_accuracy: 0.8054
32/32 [==============================] - 1s 17ms/step - loss: 0.8169 - 
accuracy: 0.8096
Epoch 5/65 - Test Accuracy: 80.96%
250/250 [==============================] - 168s 670ms/step - loss: 
0.2009 - accuracy: 0.9667 - val_loss: 0.3739 - val_accuracy: 0.9002
32/32 [==============================] - 1s 17ms/step - loss: 0.3684 - 
accuracy: 0.9131
Epoch 6/65 - Test Accuracy: 91.31%
250/250 [==============================] - 168s 670ms/step - loss: 
0.1971 - accuracy: 0.9687 - val_loss: 0.5278 - val_accuracy: 0.8569
32/32 [==============================] - 1s 17ms/step - loss: 0.5038 - 
accuracy: 0.8818
Epoch 7/65 - Test Accuracy: 88.18%
250/250 [==============================] - 167s 669ms/step - loss: 
0.2016 - accuracy: 0.9659 - val_loss: 0.6540 - val_accuracy: 0.8427
32/32 [==============================] - 1s 17ms/step - loss: 0.6595 - 
accuracy: 0.8467
Epoch 8/65 - Test Accuracy: 84.67%
250/250 [==============================] - 167s 667ms/step - loss: 
0.1980 - accuracy: 0.9657 - val_loss: 0.3382 - val_accuracy: 0.9093
32/32 [==============================] - 1s 17ms/step - loss: 0.3319 - 
accuracy: 0.9287
Epoch 9/65 - Test Accuracy: 92.87%
250/250 [==============================] - 166s 662ms/step - loss: 
0.1984 - accuracy: 0.9653 - val_loss: 0.5651 - val_accuracy: 0.8427
32/32 [==============================] - 1s 17ms/step - loss: 0.5581 - 
accuracy: 0.8545
Epoch 10/65 - Test Accuracy: 85.45%
250/250 [==============================] - 166s 664ms/step - loss: 
0.2005 - accuracy: 0.9656 - val_loss: 0.3168 - val_accuracy: 0.9123
32/32 [==============================] - 1s 17ms/step - loss: 0.3303 - 
accuracy: 0.9131
Epoch 11/65 - Test Accuracy: 91.31%
250/250 [==============================] - 167s 668ms/step - loss: 
0.1914 - accuracy: 0.9697 - val_loss: 0.3092 - val_accuracy: 0.9224
32/32 [==============================] - 1s 17ms/step - loss: 0.3111 - 
accuracy: 0.9336
Epoch 12/65 - Test Accuracy: 93.36%
250/250 [==============================] - 166s 663ms/step - loss: 
0.1843 - accuracy: 0.9699 - val_loss: 0.3247 - val_accuracy: 0.9163
32/32 [==============================] - 1s 17ms/step - loss: 0.3270 - 
```

```
accuracy: 0.9229
Epoch 13/65 - Test Accuracy: 92.29%
250/250 [==============================] - 167s 667ms/step - loss:
0.1833 - accuracy: 0.9692 - val_loss: 0.1804 - val_accuracy: 0.9728
32/32 [==============================] - 1s 17ms/step - loss: 0.1977 -
accuracy: 0.9648
Epoch 14/65 - Test Accuracy: 96.48%
250/250 [==============================] - 167s 668ms/step - loss:
0.1867 - accuracy: 0.9678 - val_loss: 0.2509 - val_accuracy: 0.9335
32/32 [==============================] - 1s 17ms/step - loss: 0.2609 -
accuracy: 0.9434
Epoch 15/65 - Test Accuracy: 94.34%
250/250 [==============================] - 168s 674ms/step - loss:
0.1798 - accuracy: 0.9707 - val_loss: 0.4485 - val_accuracy: 0.8730
32/32 [==============================] - 1s 17ms/step - loss: 0.4491 -
accuracy: 0.8809
Epoch 16/65 - Test Accuracy: 88.09%
250/250 [==============================] - 167s 667ms/step - loss:
0.1824 - accuracy: 0.9698 - val_loss: 0.2577 - val_accuracy: 0.9345
32/32 [==============================] - 1s 17ms/step - loss: 0.2296 -
accuracy: 0.9404
Epoch 17/65 - Test Accuracy: 94.04%
250/250 [==============================] - 167s 669ms/step - loss:
0.1819 - accuracy: 0.9692 - val_loss: 0.6804 - val_accuracy: 0.8266
32/32 [==============================] - 1s 17ms/step - loss: 0.6569 -
accuracy: 0.8389
Epoch 18/65 - Test Accuracy: 83.89%
250/250 [==============================] - 168s 673ms/step - loss:
0.1830 - accuracy: 0.9709 - val_loss: 0.3716 - val_accuracy: 0.9002
32/32 [==============================] - 1s 17ms/step - loss: 0.3746 -
accuracy: 0.9102
Epoch 19/65 - Test Accuracy: 91.02%
250/250 [==============================] - 170s 680ms/step - loss:
0.1747 - accuracy: 0.9721 - val_loss: 0.6807 - val_accuracy: 0.8256
32/32 [==============================] - 1s 17ms/step - loss: 0.6455 -
accuracy: 0.8438
Epoch 20/65 - Test Accuracy: 84.38%
250/250 [==============================] - 168s 670ms/step - loss:
0.1763 - accuracy: 0.9693 - val_loss: 0.2270 - val_accuracy: 0.9496
32/32 [==============================] - 1s 17ms/step - loss: 0.2493 -
accuracy: 0.9521
Epoch 21/65 - Test Accuracy: 95.21%
250/250 [==============================] - 168s 672ms/step - loss:
0.1764 - accuracy: 0.9719 - val_loss: 0.1958 - val_accuracy: 0.9627
32/32 [==============================] - 1s 17ms/step - loss: 0.1654 -
accuracy: 0.9697
Epoch 22/65 - Test Accuracy: 96.97%
250/250 [==============================] - 168s 672ms/step - loss:
0.1786 - accuracy: 0.9717 - val_loss: 0.6407 - val_accuracy: 0.8397
```

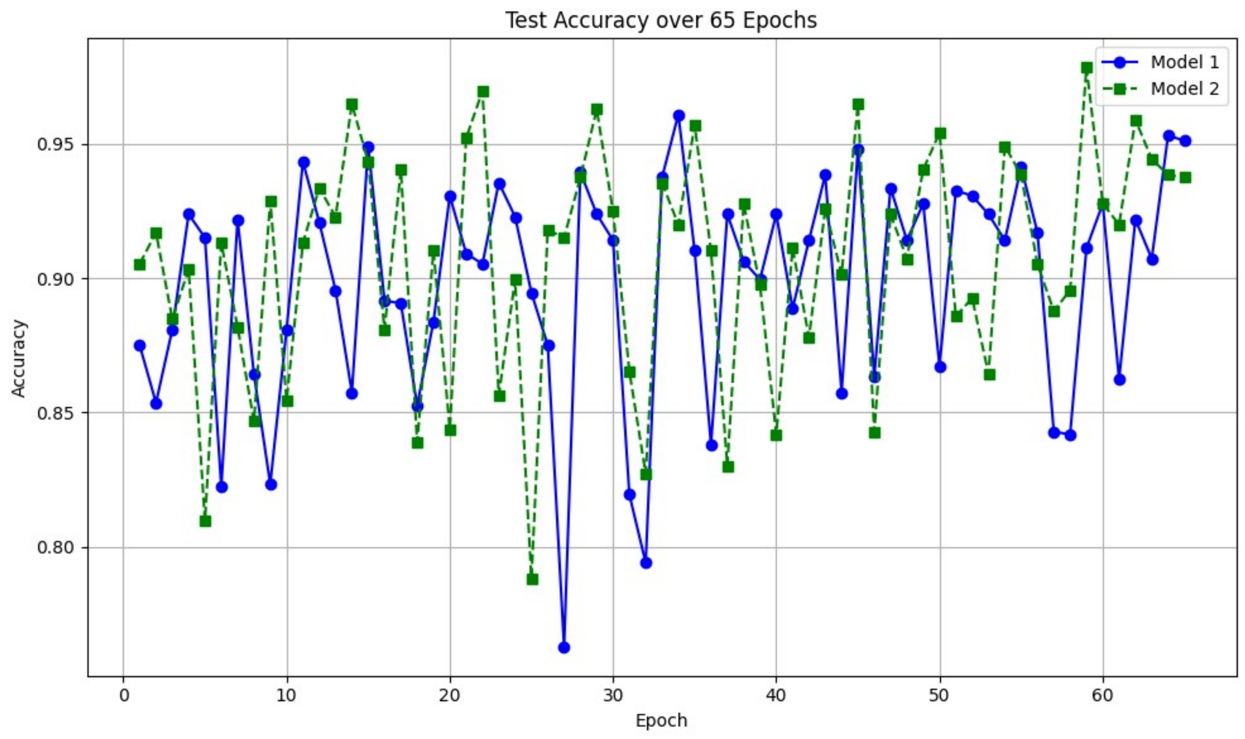
```
32/32 [==============================] - 1s 17ms/step - loss: 0.6198 -
accuracy: 0.8564
Epoch 23/65 - Test Accuracy: 85.64%
250/250 [==============================] - 170s 681ms/step - loss:
0.1729 - accuracy: 0.9707 - val_loss: 0.3914 - val_accuracy: 0.9052
32/32 [==============================] - 1s 17ms/step - loss: 0.4220 -
accuracy: 0.8994
Epoch 24/65 - Test Accuracy: 89.94%
250/250 [==============================] - 170s 679ms/step - loss:
0.1664 - accuracy: 0.9724 - val_loss: 0.8711 - val_accuracy: 0.7772
32/32 [==============================] - 1s 17ms/step - loss: 0.8590 -
accuracy: 0.7881
Epoch 25/65 - Test Accuracy: 78.81%
250/250 [==============================] - 171s 683ms/step - loss:
0.1675 - accuracy: 0.9731 - val_loss: 0.3263 - val_accuracy: 0.9143
32/32 [==============================] - 1s 17ms/step - loss: 0.3171 -
accuracy: 0.9180
Epoch 26/65 - Test Accuracy: 91.80%
250/250 [==============================] - 171s 683ms/step - loss:
0.1625 - accuracy: 0.9760 - val_loss: 0.4052 - val_accuracy: 0.8851
32/32 [==============================] - 1s 17ms/step - loss: 0.3734 -
accuracy: 0.9150
Epoch 27/65 - Test Accuracy: 91.50%
250/250 [==============================] - 171s 683ms/step - loss:
0.1580 - accuracy: 0.9752 - val_loss: 0.2579 - val_accuracy: 0.9345
32/32 [==============================] - 1s 17ms/step - loss: 0.2632 -
accuracy: 0.9375
Epoch 28/65 - Test Accuracy: 93.75%
250/250 [==============================] - 170s 679ms/step - loss:
0.1678 - accuracy: 0.9734 - val_loss: 0.2112 - val_accuracy: 0.9506
32/32 [==============================] - 1s 17ms/step - loss: 0.1957 -
accuracy: 0.9629
Epoch 29/65 - Test Accuracy: 96.29%
250/250 [==============================] - 170s 678ms/step - loss:
0.1685 - accuracy: 0.9729 - val_loss: 0.3529 - val_accuracy: 0.9113
32/32 [==============================] - 1s 17ms/step - loss: 0.3532 -
accuracy: 0.9248
Epoch 30/65 - Test Accuracy: 92.48%
250/250 [==============================] - 169s 675ms/step - loss:
0.1575 - accuracy: 0.9782 - val_loss: 0.4889 - val_accuracy: 0.8679
32/32 [==============================] - 1s 17ms/step - loss: 0.5008 -
accuracy: 0.8652
Epoch 31/65 - Test Accuracy: 86.52%
250/250 [==============================] - 170s 679ms/step - loss:
0.1672 - accuracy: 0.9743 - val_loss: 0.8736 - val_accuracy: 0.7984
32/32 [==============================] - 1s 17ms/step - loss: 0.8234 -
accuracy: 0.8271
Epoch 32/65 - Test Accuracy: 82.71%
250/250 [==============================] - 169s 676ms/step - loss:
```

```
0.1636 - accuracy: 0.9716 - val_loss: 0.3299 - val_accuracy: 0.9143
32/32 [==============================] - 1s 17ms/step - loss: 0.2735 -
accuracy: 0.9355
Epoch 33/65 - Test Accuracy: 93.55%
250/250 [==============================] - 169s 675ms/step - loss:
0.1547 - accuracy: 0.9762 - val_loss: 0.3268 - val_accuracy: 0.9113
32/32 [==============================] - 1s 17ms/step - loss: 0.3275 -
accuracy: 0.9199
Epoch 34/65 - Test Accuracy: 91.99%
250/250 [==============================] - 167s 669ms/step - loss:
0.1512 - accuracy: 0.9787 - val_loss: 0.2252 - val_accuracy: 0.9526
32/32 [==============================] - 1s 17ms/step - loss: 0.2011 -
accuracy: 0.9570
Epoch 35/65 - Test Accuracy: 95.70%
250/250 [==============================] - 169s 676ms/step - loss:
0.1603 - accuracy: 0.9736 - val_loss: 0.4362 - val_accuracy: 0.8851
32/32 [==============================] - 1s 17ms/step - loss: 0.3798 -
accuracy: 0.9102
Epoch 36/65 - Test Accuracy: 91.02%
250/250 [==============================] - 167s 670ms/step - loss:
0.1601 - accuracy: 0.9729 - val_loss: 0.7854 - val_accuracy: 0.8165
32/32 [==============================] - 1s 17ms/step - loss: 0.7492 -
accuracy: 0.8301
Epoch 37/65 - Test Accuracy: 83.01%
250/250 [==============================] - 168s 672ms/step - loss:
0.1526 - accuracy: 0.9768 - val_loss: 0.3940 - val_accuracy: 0.9022
32/32 [==============================] - 1s 17ms/step - loss: 0.3323 -
accuracy: 0.9277
Epoch 38/65 - Test Accuracy: 92.77%
250/250 [==============================] - 168s 672ms/step - loss:
0.1545 - accuracy: 0.9788 - val_loss: 0.4306 - val_accuracy: 0.8861
32/32 [==============================] - 1s 17ms/step - loss: 0.4277 -
accuracy: 0.8975
Epoch 39/65 - Test Accuracy: 89.75%
250/250 [==============================] - 169s 674ms/step - loss:
0.1525 - accuracy: 0.9743 - val_loss: 0.5488 - val_accuracy: 0.8569
32/32 [==============================] - 1s 17ms/step - loss: 0.6077 -
accuracy: 0.8418
Epoch 40/65 - Test Accuracy: 84.18%
250/250 [==============================] - 169s 674ms/step - loss:
0.1475 - accuracy: 0.9781 - val_loss: 0.3421 - val_accuracy: 0.9173
32/32 [==============================] - 1s 17ms/step - loss: 0.3950 -
accuracy: 0.9111
Epoch 41/65 - Test Accuracy: 91.11%
250/250 [==============================] - 170s 680ms/step - loss:
0.1482 - accuracy: 0.9776 - val_loss: 0.5175 - val_accuracy: 0.8639
32/32 [==============================] - 1s 17ms/step - loss: 0.5211 -
accuracy: 0.8779
Epoch 42/65 - Test Accuracy: 87.79%
```

```
250/250 [==============================] - 169s 677ms/step - loss:
0.1497 - accuracy: 0.9772 - val_loss: 0.3206 - val_accuracy: 0.9224
32/32 [==============================] - 1s 17ms/step - loss: 0.2779 -
accuracy: 0.9258
Epoch 43/65 - Test Accuracy: 92.58%
250/250 [==============================] - 170s 679ms/step - loss:
0.1391 - accuracy: 0.9815 - val_loss: 0.3953 - val_accuracy: 0.8972
32/32 [==============================] - 1s 17ms/step - loss: 0.3907 -
accuracy: 0.9014
Epoch 44/65 - Test Accuracy: 90.14%
250/250 [==============================] - 170s 680ms/step - loss:
0.1497 - accuracy: 0.9768 - val_loss: 0.1687 - val_accuracy: 0.9637
32/32 [==============================] - 1s 17ms/step - loss: 0.1720 -
accuracy: 0.9648
Epoch 45/65 - Test Accuracy: 96.48%
250/250 [==============================] - 168s 672ms/step - loss:
0.1454 - accuracy: 0.9782 - val_loss: 0.5971 - val_accuracy: 0.8246
32/32 [==============================] - 1s 17ms/step - loss: 0.5625 -
accuracy: 0.8428
Epoch 46/65 - Test Accuracy: 84.28%
250/250 [==============================] - 168s 671ms/step - loss:
0.1441 - accuracy: 0.9777 - val_loss: 0.3025 - val_accuracy: 0.9194
32/32 [==============================] - 1s 17ms/step - loss: 0.3086 -
accuracy: 0.9238
Epoch 47/65 - Test Accuracy: 92.38%
250/250 [==============================] - 167s 669ms/step - loss:
0.1479 - accuracy: 0.9763 - val_loss: 0.3697 - val_accuracy: 0.9052
32/32 [==============================] - 1s 17ms/step - loss: 0.3928 -
accuracy: 0.9072
Epoch 48/65 - Test Accuracy: 90.72%
250/250 [==============================] - 170s 678ms/step - loss:
0.1481 - accuracy: 0.9761 - val_loss: 0.2660 - val_accuracy: 0.9294
32/32 [==============================] - 1s 17ms/step - loss: 0.2844 -
accuracy: 0.9404
Epoch 49/65 - Test Accuracy: 94.04%
250/250 [==============================] - 167s 668ms/step - loss:
0.1544 - accuracy: 0.9744 - val_loss: 0.2330 - val_accuracy: 0.9425
32/32 [==============================] - 1s 17ms/step - loss: 0.2076 -
accuracy: 0.9541
Epoch 50/65 - Test Accuracy: 95.41%
250/250 [==============================] - 167s 667ms/step - loss:
0.1405 - accuracy: 0.9781 - val_loss: 0.5458 - val_accuracy: 0.8659
32/32 [==============================] - 1s 17ms/step - loss: 0.4267 -
accuracy: 0.8857
Epoch 51/65 - Test Accuracy: 88.57%
250/250 [==============================] - 165s 662ms/step - loss:
0.1439 - accuracy: 0.9777 - val_loss: 0.4594 - val_accuracy: 0.8851
32/32 [==============================] - 1s 17ms/step - loss: 0.4477 -
accuracy: 0.8926
```

```
Epoch 52/65 - Test Accuracy: 89.26%
250/250 [==============================] - 167s 668ms/step - loss:
0.1465 - accuracy: 0.9761 - val_loss: 0.4408 - val_accuracy: 0.8730
32/32 [==============================] - 1s 17ms/step - loss: 0.4696 -
accuracy: 0.8643
Epoch 53/65 - Test Accuracy: 86.43%
250/250 [==============================] - 167s 668ms/step - loss:
0.1447 - accuracy: 0.9772 - val_loss: 0.2462 - val_accuracy: 0.9435
32/32 [==============================] - 1s 17ms/step - loss: 0.2482 -
accuracy: 0.9492
Epoch 54/65 - Test Accuracy: 94.92%
250/250 [==============================] - 166s 664ms/step - loss:
0.1465 - accuracy: 0.9772 - val_loss: 0.2864 - val_accuracy: 0.9254
32/32 [==============================] - 1s 17ms/step - loss: 0.2816 -
accuracy: 0.9385
Epoch 55/65 - Test Accuracy: 93.85%
250/250 [==============================] - 165s 661ms/step - loss:
0.1466 - accuracy: 0.9768 - val_loss: 0.3595 - val_accuracy: 0.9052
32/32 [==============================] - 1s 17ms/step - loss: 0.3652 -
accuracy: 0.9053
Epoch 56/65 - Test Accuracy: 90.53%
250/250 [==============================] - 167s 667ms/step - loss:
0.1364 - accuracy: 0.9800 - val_loss: 0.3906 - val_accuracy: 0.8942
32/32 [==============================] - 1s 17ms/step - loss: 0.4346 -
accuracy: 0.8877
Epoch 57/65 - Test Accuracy: 88.77%
250/250 [==============================] - 167s 669ms/step - loss:
0.1356 - accuracy: 0.9792 - val_loss: 0.4682 - val_accuracy: 0.8871
32/32 [==============================] - 1s 17ms/step - loss: 0.4726 -
accuracy: 0.8955
Epoch 58/65 - Test Accuracy: 89.55%
250/250 [==============================] - 168s 672ms/step - loss:
0.1363 - accuracy: 0.9792 - val_loss: 0.1529 - val_accuracy: 0.9718
32/32 [==============================] - 1s 17ms/step - loss: 0.1280 -
accuracy: 0.9785
Epoch 59/65 - Test Accuracy: 97.85%
250/250 [==============================] - 167s 668ms/step - loss:
0.1383 - accuracy: 0.9780 - val_loss: 0.2648 - val_accuracy: 0.9315
32/32 [==============================] - 1s 17ms/step - loss: 0.2804 -
accuracy: 0.9277
Epoch 60/65 - Test Accuracy: 92.77%
250/250 [==============================] - 168s 670ms/step - loss:
0.1323 - accuracy: 0.9823 - val_loss: 0.3441 - val_accuracy: 0.9194
32/32 [==============================] - 1s 17ms/step - loss: 0.3490 -
accuracy: 0.9199
Epoch 61/65 - Test Accuracy: 91.99%
250/250 [==============================] - 167s 667ms/step - loss:
0.1420 - accuracy: 0.9753 - val_loss: 0.1702 - val_accuracy: 0.9637
32/32 [==============================] - 1s 17ms/step - loss: 0.1852 -
```

```
accuracy: 0.9590
Epoch 62/65 - Test Accuracy: 95.90%
250/250 [==============================] - 168s 670ms/step - loss:
0.1356 - accuracy: 0.9785 - val_loss: 0.2315 - val_accuracy: 0.9425
32/32 [==============================] - 1s 17ms/step - loss: 0.2532 -
accuracy: 0.9443
Epoch 63/65 - Test Accuracy: 94.43%
250/250 [==============================] - 167s 667ms/step - loss:
0.1466 - accuracy: 0.9727 - val_loss: 0.2522 - val_accuracy: 0.9425
32/32 [==============================] - 1s 17ms/step - loss: 0.2570 -
accuracy: 0.9385
Epoch 64/65 - Test Accuracy: 93.85%
250/250 [==============================] - 166s 666ms/step - loss:
0.1283 - accuracy: 0.9835 - val_loss: 0.3268 - val_accuracy: 0.9163
32/32 [==============================] - 1s 17ms/step - loss: 0.2654 -
accuracy: 0.9375
Epoch 65/65 - Test Accuracy: 93.75%

# Calculate the mean test accuracy for model 2:
mean_test_accuracy_2 = np.mean(test_accuracy_2)
print(f'Mean Test Accuracy for model 2 over {epochs} epochs:
{mean_test_accuracy_2 * 100:.2f}%')

Mean Test Accuracy for model 2 over 65 epochs: 93.75%

# Plot the test accuracy over epochs for 2 models and comparision
plt.figure(figsize=(10, 6))
plt.plot(range(1, epochs + 1), test_accuracy_history, label='Model 1',
marker='o', linestyle='-', color='blue')
plt.plot(range(1, epochs + 1), test_accuracy_history_2, label='Model
2', marker='s', linestyle='--', color='green')

plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Test Accuracy over 65 Epochs')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



After model creation with 65 epochs,we realized that the early stopping should be conducted with the decrease value of patience from 20 to 10 to 5 .

```
# Changing the patience parameter to 5
from keras.callbacks import EarlyStopping
calls_2 = EarlyStopping(monitor = 'val_accuracy',
                   mode = 'max',
                   patience = 3,# stop if not improve for 3
consecutive epochs
                   verbose = 1,
                   restore_best_weights = True)

import numpy as np

#Test accuracy using Early stopping
test_accuracy_history_2 = []
epochs = 65
for epoch in range(epochs):
    history_2 = model_2.fit(
        train_ds,
        batch_size=BATCH_SIZE,
        validation_data=val_ds,
        verbose=1,
        epochs=1,
        callbacks=[calls_2]  # Add the EarlyStopping callback
    )

    test_loss_2, test_accuracy_2 = model_2.evaluate(test_ds)
    print(f'Epoch {epoch + 1}/{epochs} - Test Accuracy:
{test_accuracy_2 * 100:.2f}%')

    test_accuracy_history_2.append(test_accuracy_2)

    if calls_2.stopped_epoch > 0:
        print(f"Early stopping at epoch {epoch + 1} due to no
improvement in validation accuracy.")
        break  # Break the loop if early stopping criteria met

# Calculate the mean test accuracy
mean_test_accuracy_2 = np.mean(test_accuracy_history_2)
print(f'Mean Test Accuracy over {epoch + 1} epochs:
{mean_test_accuracy_2 * 100:.2f}%')

250/250 [==============================] - 172s 687ms/step - loss:
0.2079 - accuracy: 0.9642 - val_loss: 0.2791 - val_accuracy: 0.9345
32/32 [==============================] - 1s 26ms/step - loss: 0.2328 -
accuracy: 0.9521
Epoch 1/65 - Test Accuracy: 95.21%
250/250 [==============================] - 172s 686ms/step - loss:
0.2009 - accuracy: 0.9653 - val_loss: 0.6828 - val_accuracy: 0.8458
32/32 [==============================] - 1s 26ms/step - loss: 0.6726 -
accuracy: 0.8379
Epoch 2/65 - Test Accuracy: 83.79%
250/250 [==============================] - 165s 662ms/step - loss:
0.1997 - accuracy: 0.9673 - val_loss: 0.3171 - val_accuracy: 0.9325
32/32 [==============================] - 1s 26ms/step - loss: 0.2950 -
accuracy: 0.9404
Epoch 3/65 - Test Accuracy: 94.04%
250/250 [==============================] - 166s 665ms/step - loss:
0.2030 - accuracy: 0.9652 - val_loss: 0.4213 - val_accuracy: 0.9012
32/32 [==============================] - 1s 26ms/step - loss: 0.3857 -
accuracy: 0.9023
Epoch 4/65 - Test Accuracy: 90.23%
250/250 [==============================] - 166s 662ms/step - loss:
0.2016 - accuracy: 0.9658 - val_loss: 0.3268 - val_accuracy: 0.9315
32/32 [==============================] - 1s 26ms/step - loss: 0.3097 -
accuracy: 0.9316
Epoch 5/65 - Test Accuracy: 93.16%
250/250 [==============================] - 166s 662ms/step - loss:
0.1920 - accuracy: 0.9677 - val_loss: 0.4199 - val_accuracy: 0.8931
32/32 [==============================] - 1s 26ms/step - loss: 0.3765 -
accuracy: 0.8936
Epoch 6/65 - Test Accuracy: 89.36%
250/250 [==============================] - 166s 663ms/step - loss:
0.1846 - accuracy: 0.9706 - val_loss: 0.4147 - val_accuracy: 0.8911
32/32 [==============================] - 1s 26ms/step - loss: 0.3822 -
accuracy: 0.8965
Epoch 7/65 - Test Accuracy: 89.65%
```

```
250/250 [==============================] - 170s 682ms/step - loss:
0.1774 - accuracy: 0.9728 - val_loss: 0.3058 - val_accuracy: 0.9345
32/32 [==============================] - 1s 26ms/step - loss: 0.2637 -
accuracy: 0.9365
Epoch 8/65 - Test Accuracy: 93.65%
250/250 [==============================] - 168s 674ms/step - loss:
0.1846 - accuracy: 0.9689 - val_loss: 0.7800 - val_accuracy: 0.7994
32/32 [==============================] - 1s 26ms/step - loss: 0.7735 -
accuracy: 0.7949
Epoch 9/65 - Test Accuracy: 79.49%
250/250 [==============================] - 167s 670ms/step - loss:
0.1900 - accuracy: 0.9669 - val_loss: 0.3351 - val_accuracy: 0.9274
32/32 [==============================] - 1s 26ms/step - loss: 0.2724 -
accuracy: 0.9316
Epoch 10/65 - Test Accuracy: 93.16%
250/250 [==============================] - 168s 671ms/step - loss:
0.1839 - accuracy: 0.9687 - val_loss: 0.2827 - val_accuracy: 0.9375
32/32 [==============================] - 1s 26ms/step - loss: 0.2379 -
accuracy: 0.9414
Epoch 11/65 - Test Accuracy: 94.14%
250/250 [==============================] - 167s 666ms/step - loss:
0.1847 - accuracy: 0.9694 - val_loss: 0.4328 - val_accuracy: 0.8921
32/32 [==============================] - 1s 26ms/step - loss: 0.4013 -
accuracy: 0.9023
Epoch 12/65 - Test Accuracy: 90.23%
250/250 [==============================] - 166s 666ms/step - loss:
0.1891 - accuracy: 0.9706 - val_loss: 0.8675 - val_accuracy: 0.7853
32/32 [==============================] - 1s 26ms/step - loss: 0.8566 -
accuracy: 0.7734
Epoch 13/65 - Test Accuracy: 77.34%
250/250 [==============================] - 165s 661ms/step - loss:
0.1854 - accuracy: 0.9704 - val_loss: 0.2861 - val_accuracy: 0.9385
32/32 [==============================] - 1s 26ms/step - loss: 0.2486 -
accuracy: 0.9414
Epoch 14/65 - Test Accuracy: 94.14%
250/250 [==============================] - 165s 659ms/step - loss:
0.1842 - accuracy: 0.9696 - val_loss: 0.8578 - val_accuracy: 0.7873
32/32 [==============================] - 1s 26ms/step - loss: 0.8364 -
accuracy: 0.7988
Epoch 15/65 - Test Accuracy: 79.88%
250/250 [==============================] - 163s 654ms/step - loss:
0.1837 - accuracy: 0.9694 - val_loss: 0.4330 - val_accuracy: 0.8871
32/32 [==============================] - 1s 26ms/step - loss: 0.3696 -
accuracy: 0.9043
Epoch 16/65 - Test Accuracy: 90.43%
250/250 [==============================] - 163s 652ms/step - loss:
0.1736 - accuracy: 0.9722 - val_loss: 0.3111 - val_accuracy: 0.9274
32/32 [==============================] - 1s 26ms/step - loss: 0.2917 -
accuracy: 0.9258
```

```
Epoch 17/65 - Test Accuracy: 92.58%
250/250 [==============================] - 162s 648ms/step - loss:
0.1755 - accuracy: 0.9702 - val_loss: 0.4433 - val_accuracy: 0.8861
32/32 [==============================] - 1s 26ms/step - loss: 0.4003 -
accuracy: 0.8936
Epoch 18/65 - Test Accuracy: 89.36%
250/250 [==============================] - 163s 652ms/step - loss:
0.1734 - accuracy: 0.9723 - val_loss: 0.5340 - val_accuracy: 0.8710
32/32 [==============================] - 1s 26ms/step - loss: 0.4636 -
accuracy: 0.8789
Epoch 19/65 - Test Accuracy: 87.89%
250/250 [==============================] - 162s 649ms/step - loss:
0.1588 - accuracy: 0.9772 - val_loss: 0.4865 - val_accuracy: 0.8730
32/32 [==============================] - 1s 26ms/step - loss: 0.4256 -
accuracy: 0.8857
Epoch 20/65 - Test Accuracy: 88.57%
250/250 [==============================] - 164s 654ms/step - loss:
0.1720 - accuracy: 0.9713 - val_loss: 0.3332 - val_accuracy: 0.9073
32/32 [==============================] - 1s 26ms/step - loss: 0.3403 -
accuracy: 0.9102
Epoch 21/65 - Test Accuracy: 91.02%
250/250 [==============================] - 165s 662ms/step - loss:
0.1691 - accuracy: 0.9724 - val_loss: 0.3128 - val_accuracy: 0.9284
32/32 [==============================] - 1s 26ms/step - loss: 0.2952 -
accuracy: 0.9297
Epoch 22/65 - Test Accuracy: 92.97%
250/250 [==============================] - 165s 660ms/step - loss:
0.1687 - accuracy: 0.9733 - val_loss: 0.4472 - val_accuracy: 0.8831
32/32 [==============================] - 1s 26ms/step - loss: 0.4069 -
accuracy: 0.8799
Epoch 23/65 - Test Accuracy: 87.99%
250/250 [==============================] - 164s 656ms/step - loss:
0.1638 - accuracy: 0.9741 - val_loss: 0.3751 - val_accuracy: 0.9163
32/32 [==============================] - 1s 26ms/step - loss: 0.2910 -
accuracy: 0.9297
Epoch 24/65 - Test Accuracy: 92.97%
250/250 [==============================] - 164s 656ms/step - loss:
0.1579 - accuracy: 0.9752 - val_loss: 0.3654 - val_accuracy: 0.9173
32/32 [==============================] - 1s 26ms/step - loss: 0.3200 -
accuracy: 0.9209
Epoch 25/65 - Test Accuracy: 92.09%
250/250 [==============================] - 163s 653ms/step - loss:
0.1681 - accuracy: 0.9713 - val_loss: 0.5685 - val_accuracy: 0.8720
32/32 [==============================] - 1s 26ms/step - loss: 0.5002 -
accuracy: 0.8623
Epoch 26/65 - Test Accuracy: 86.23%
250/250 [==============================] - 165s 659ms/step - loss:
0.1720 - accuracy: 0.9712 - val_loss: 0.2346 - val_accuracy: 0.9526
32/32 [==============================] - 1s 26ms/step - loss: 0.2024 -
```

```
accuracy: 0.9590
Epoch 27/65 - Test Accuracy: 95.90%
250/250 [==============================] - 165s 658ms/step - loss:
0.1633 - accuracy: 0.9723 - val_loss: 0.3073 - val_accuracy: 0.9214
32/32 [==============================] - 1s 26ms/step - loss: 0.2947 -
accuracy: 0.9229
Epoch 28/65 - Test Accuracy: 92.29%
250/250 [==============================] - 164s 657ms/step - loss:
0.1516 - accuracy: 0.9762 - val_loss: 0.6666 - val_accuracy: 0.8458
32/32 [==============================] - 1s 27ms/step - loss: 0.5959 -
accuracy: 0.8438
Epoch 29/65 - Test Accuracy: 84.38%
250/250 [==============================] - 165s 658ms/step - loss:
0.1655 - accuracy: 0.9736 - val_loss: 0.3085 - val_accuracy: 0.9254
32/32 [==============================] - 1s 26ms/step - loss: 0.2963 -
accuracy: 0.9209
Epoch 30/65 - Test Accuracy: 92.09%
250/250 [==============================] - 164s 656ms/step - loss:
0.1529 - accuracy: 0.9781 - val_loss: 0.4668 - val_accuracy: 0.8760
32/32 [==============================] - 1s 26ms/step - loss: 0.4014 -
accuracy: 0.8936
Epoch 31/65 - Test Accuracy: 89.36%
250/250 [==============================] - 162s 647ms/step - loss:
0.1547 - accuracy: 0.9757 - val_loss: 0.2843 - val_accuracy: 0.9345
32/32 [==============================] - 1s 26ms/step - loss: 0.2440 -
accuracy: 0.9434
Epoch 32/65 - Test Accuracy: 94.34%
250/250 [==============================] - 163s 652ms/step - loss:
0.1563 - accuracy: 0.9748 - val_loss: 0.3521 - val_accuracy: 0.9194
32/32 [==============================] - 1s 26ms/step - loss: 0.2744 -
accuracy: 0.9453
Epoch 33/65 - Test Accuracy: 94.53%
250/250 [==============================] - 163s 652ms/step - loss:
0.1593 - accuracy: 0.9746 - val_loss: 0.3949 - val_accuracy: 0.8931
32/32 [==============================] - 1s 26ms/step - loss: 0.3548 -
accuracy: 0.9004
Epoch 34/65 - Test Accuracy: 90.04%
250/250 [==============================] - 163s 652ms/step - loss:
0.1523 - accuracy: 0.9757 - val_loss: 0.5894 - val_accuracy: 0.8528
32/32 [==============================] - 1s 26ms/step - loss: 0.5083 -
accuracy: 0.8584
Epoch 35/65 - Test Accuracy: 85.84%
250/250 [==============================] - 163s 652ms/step - loss:
0.1504 - accuracy: 0.9761 - val_loss: 0.2770 - val_accuracy: 0.9375
32/32 [==============================] - 1s 26ms/step - loss: 0.2596 -
accuracy: 0.9482
Epoch 36/65 - Test Accuracy: 94.82%
250/250 [==============================] - 164s 654ms/step - loss:
0.1507 - accuracy: 0.9766 - val_loss: 0.6582 - val_accuracy: 0.8236
```

```
32/32 [==============================] - 1s 26ms/step - loss: 0.6187 -
accuracy: 0.8311
Epoch 37/65 - Test Accuracy: 83.11%
250/250 [==============================] - 163s 654ms/step - loss:
0.1549 - accuracy: 0.9749 - val_loss: 0.6424 - val_accuracy: 0.8538
32/32 [==============================] - 1s 26ms/step - loss: 0.6162 -
accuracy: 0.8486
Epoch 38/65 - Test Accuracy: 84.86%
250/250 [==============================] - 162s 647ms/step - loss:
0.1460 - accuracy: 0.9778 - val_loss: 0.3915 - val_accuracy: 0.9032
32/32 [==============================] - 1s 26ms/step - loss: 0.3547 -
accuracy: 0.9131
Epoch 39/65 - Test Accuracy: 91.31%
250/250 [==============================] - 163s 651ms/step - loss:
0.1507 - accuracy: 0.9752 - val_loss: 0.2937 - val_accuracy: 0.9294
32/32 [==============================] - 1s 26ms/step - loss: 0.2870 -
accuracy: 0.9316
Epoch 40/65 - Test Accuracy: 93.16%
250/250 [==============================] - 163s 653ms/step - loss:
0.1506 - accuracy: 0.9781 - val_loss: 0.2951 - val_accuracy: 0.9315
32/32 [==============================] - 1s 26ms/step - loss: 0.2621 -
accuracy: 0.9385
Epoch 41/65 - Test Accuracy: 93.85%
250/250 [==============================] - 163s 650ms/step - loss:
0.1468 - accuracy: 0.9791 - val_loss: 0.3585 - val_accuracy: 0.9173
32/32 [==============================] - 1s 26ms/step - loss: 0.3087 -
accuracy: 0.9219
Epoch 42/65 - Test Accuracy: 92.19%
250/250 [==============================] - 163s 651ms/step - loss:
0.1583 - accuracy: 0.9739 - val_loss: 0.4171 - val_accuracy: 0.9093
32/32 [==============================] - 1s 26ms/step - loss: 0.3799 -
accuracy: 0.9111
Epoch 43/65 - Test Accuracy: 91.11%
250/250 [==============================] - 163s 650ms/step - loss:
0.1551 - accuracy: 0.9753 - val_loss: 0.2799 - val_accuracy: 0.9335
32/32 [==============================] - 1s 26ms/step - loss: 0.2178 -
accuracy: 0.9492
Epoch 44/65 - Test Accuracy: 94.92%
250/250 [==============================] - 163s 651ms/step - loss:
0.1461 - accuracy: 0.9767 - val_loss: 0.2953 - val_accuracy: 0.9345
32/32 [==============================] - 1s 26ms/step - loss: 0.2490 -
accuracy: 0.9385
Epoch 45/65 - Test Accuracy: 93.85%
250/250 [==============================] - 161s 643ms/step - loss:
0.1452 - accuracy: 0.9778 - val_loss: 0.3944 - val_accuracy: 0.9032
32/32 [==============================] - 1s 26ms/step - loss: 0.3207 -
accuracy: 0.9268
Epoch 46/65 - Test Accuracy: 92.68%
250/250 [==============================] - 162s 648ms/step - loss:
```

```
0.1419 - accuracy: 0.9767 - val_loss: 0.2378 - val_accuracy: 0.9405
32/32 [==============================] - 1s 26ms/step - loss: 0.2226 -
accuracy: 0.9434
Epoch 47/65 - Test Accuracy: 94.34%
250/250 [==============================] - 163s 652ms/step - loss:
0.1586 - accuracy: 0.9711 - val_loss: 0.4604 - val_accuracy: 0.9042
32/32 [==============================] - 1s 26ms/step - loss: 0.3556 -
accuracy: 0.9209
Epoch 48/65 - Test Accuracy: 92.09%
250/250 [==============================] - 163s 653ms/step - loss:
0.1395 - accuracy: 0.9796 - val_loss: 0.3698 - val_accuracy: 0.9143
32/32 [==============================] - 1s 26ms/step - loss: 0.3194 -
accuracy: 0.9219
Epoch 49/65 - Test Accuracy: 92.19%
250/250 [==============================] - 163s 650ms/step - loss:
0.1352 - accuracy: 0.9802 - val_loss: 0.5310 - val_accuracy: 0.8629
32/32 [==============================] - 1s 26ms/step - loss: 0.4655 -
accuracy: 0.8818
Epoch 50/65 - Test Accuracy: 88.18%
250/250 [==============================] - 163s 654ms/step - loss:
0.1505 - accuracy: 0.9770 - val_loss: 0.4945 - val_accuracy: 0.8800
32/32 [==============================] - 1s 26ms/step - loss: 0.4205 -
accuracy: 0.8926
Epoch 51/65 - Test Accuracy: 89.26%
250/250 [==============================] - 163s 651ms/step - loss:
0.1346 - accuracy: 0.9798 - val_loss: 0.3761 - val_accuracy: 0.9153
32/32 [==============================] - 1s 26ms/step - loss: 0.3058 -
accuracy: 0.9346
Epoch 52/65 - Test Accuracy: 93.46%
250/250 [==============================] - 162s 650ms/step - loss:
0.1453 - accuracy: 0.9752 - val_loss: 0.6015 - val_accuracy: 0.8679
32/32 [==============================] - 1s 26ms/step - loss: 0.5280 -
accuracy: 0.8730
Epoch 53/65 - Test Accuracy: 87.30%
250/250 [==============================] - 162s 647ms/step - loss:
0.1437 - accuracy: 0.9766 - val_loss: 0.4962 - val_accuracy: 0.8921
32/32 [==============================] - 1s 26ms/step - loss: 0.4036 -
accuracy: 0.9043
Epoch 54/65 - Test Accuracy: 90.43%
250/250 [==============================] - 162s 650ms/step - loss:
0.1439 - accuracy: 0.9780 - val_loss: 0.3709 - val_accuracy: 0.9093
32/32 [==============================] - 1s 26ms/step - loss: 0.3376 -
accuracy: 0.9219
Epoch 55/65 - Test Accuracy: 92.19%
250/250 [==============================] - 163s 652ms/step - loss:
0.1401 - accuracy: 0.9772 - val_loss: 0.6356 - val_accuracy: 0.8478
32/32 [==============================] - 1s 26ms/step - loss: 0.5999 -
accuracy: 0.8447
Epoch 56/65 - Test Accuracy: 84.47%
```

```
250/250 [==============================] - 163s 653ms/step - loss:
0.1332 - accuracy: 0.9821 - val_loss: 0.2247 - val_accuracy: 0.9435
32/32 [==============================] - 1s 26ms/step - loss: 0.1834 -
accuracy: 0.9619
Epoch 57/65 - Test Accuracy: 96.19%
250/250 [==============================] - 163s 653ms/step - loss:
0.1278 - accuracy: 0.9830 - val_loss: 0.4208 - val_accuracy: 0.8962
32/32 [==============================] - 1s 26ms/step - loss: 0.3639 -
accuracy: 0.9053
Epoch 58/65 - Test Accuracy: 90.53%
250/250 [==============================] - 164s 656ms/step - loss:
0.1301 - accuracy: 0.9820 - val_loss: 0.4143 - val_accuracy: 0.8992
32/32 [==============================] - 1s 26ms/step - loss: 0.3476 -
accuracy: 0.9238
Epoch 59/65 - Test Accuracy: 92.38%
250/250 [==============================] - 162s 647ms/step - loss:
0.1321 - accuracy: 0.9792 - val_loss: 0.5055 - val_accuracy: 0.8831
32/32 [==============================] - 1s 26ms/step - loss: 0.5232 -
accuracy: 0.8818
Epoch 60/65 - Test Accuracy: 88.18%
250/250 [==============================] - 163s 652ms/step - loss:
0.1288 - accuracy: 0.9833 - val_loss: 0.3300 - val_accuracy: 0.9133
32/32 [==============================] - 1s 26ms/step - loss: 0.2626 -
accuracy: 0.9346
Epoch 61/65 - Test Accuracy: 93.46%
250/250 [==============================] - 164s 656ms/step - loss:
0.1282 - accuracy: 0.9818 - val_loss: 0.2612 - val_accuracy: 0.9315
32/32 [==============================] - 1s 26ms/step - loss: 0.2181 -
accuracy: 0.9512
Epoch 62/65 - Test Accuracy: 95.12%
250/250 [==============================] - 164s 657ms/step - loss:
0.1338 - accuracy: 0.9783 - val_loss: 0.3606 - val_accuracy: 0.9214
32/32 [==============================] - 1s 26ms/step - loss: 0.2712 -
accuracy: 0.9395
Epoch 63/65 - Test Accuracy: 93.95%
250/250 [==============================] - 165s 658ms/step - loss:
0.1256 - accuracy: 0.9835 - val_loss: 0.5820 - val_accuracy: 0.8438
32/32 [==============================] - 1s 26ms/step - loss: 0.5494 -
accuracy: 0.8662
Epoch 64/65 - Test Accuracy: 86.62%
250/250 [==============================] - 164s 655ms/step - loss:
0.1297 - accuracy: 0.9811 - val_loss: 0.3336 - val_accuracy: 0.9143
32/32 [==============================] - 1s 26ms/step - loss: 0.2948 -
accuracy: 0.9355
Epoch 65/65 - Test Accuracy: 93.55%
Mean Test Accuracy over 65 epochs: 90.59%
```

Save model

```
#import os
#model_version=max([int(i) for i in os.listdir("../models") + [0]])+1
#model.save(f"../models/{model_version}")

#model.save("../tomatoes.h5")

#save the files :
from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

!cp -r /content/tomatoleaf /content/drive/My\ Drive/


!ls /content/drive/My\ Drive/
```